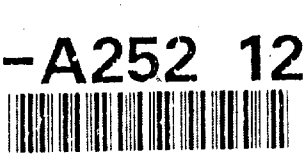


A Performance-Based Comparison of Object-Oriented Simulation Tools

MTR92B0000051
April 1992

Edward H. Bensley
Victor T. Giddings
Jonathan I. Leivent
Ronald J. Watro

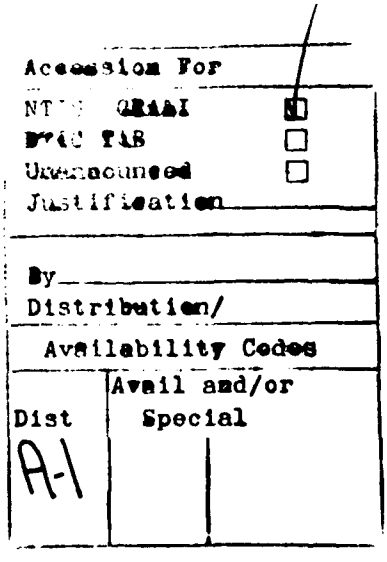
MITRE
Bedford, Massachusetts

A Performance-Based Comparison of Object-Oriented Simulation Tools

MTR92B0000051
April 1992

Edward H. Bensley
Victor T. Giddings
Jonathan I. Leivent
Ronald J. Watro

Contract Sponsor MSR
Contract No. N/A
Project No. 91330
Dept. D070



MITRE
Bedford, Massachusetts

Department Approval: Muriel J. Brooks

Muriel J. Brooks

MITRE Project Approval: Edward H. Bensley

Edward H. Bensley

ABSTRACT

This paper compares the performance and features of five different tools for object-oriented simulation. Three of the tools (MODSIM II, SES/*workbench*, and *Sim++*) are commercial products that are targeted exclusively at simulation work. Also examined are simulations in Smalltalk-80 and our own, non-commercial C++ simulation library, called MOOSE (MITRE Object-Oriented Simulation Executive). For each of the tools, we discuss the support for simulation, the support for object-oriented design and the degree to which these areas are effectively integrated. We report the results of performance testing of the tools using six concise benchmarks, each devised to test a specific feature, and one larger simulation, devised to compare general performance. Also included are partial results on ERIC, an object-oriented simulation tool developed at Rome Laboratories.

ACKNOWLEDGMENTS

This paper was presented at the 1992 Object-Oriented Simulation Conference, part of the Society for Computer Simulation (SCS) Western Simulation Multiconference, held 20-22 January in Newport Beach, CA. A condensed version of the paper appears in the conference proceedings [Bensley 92].

The Sim++ code for one of our benchmarks (the bank simulation) was written by Brett Cui under support from the Software Engineering Core Project.

The following trademarks are used throughout the remainder of the document:

- Butterfly is a trademark of BBN Advanced Computers, Inc.
- Computing Surface is a trademark of Meiko Scientific Corporation.
- Jade, *Sim++*, and TimeWarp are trademarks of Jade Simulations International Corporation.
- MODSIM II and SIMGRAPHICS II are trademarks and service marks of CACI Products Company.
- Sun-3 and Sun-4 are trademarks of Sun Microsystems, Inc.
- Smalltalk-80 and Objectworks are trademarks of ParcPlace Systems, Inc.
- SES/*workbench*, SES/*design* and SES/*sim* are trademarks of Scientific and Engineering Software, Inc.

TABLE OF CONTENTS

LIST OF FIGURES

SECTION 1

INTRODUCTION

This paper compares the performance and features of six different tools for object-oriented simulation. Three of the tools (MODSIM II, SES/*workbench*, and *Sim++*) are commercial products that are targeted at simulation work. We also examine simulations in Smalltalk-80, ERIC, and our own, non-commercial C++ simulation library, called MOOSE (MITRE Object-Oriented Simulation Executive). MOOSE is included to represent a simulation system coded quickly (in less than two staff months) using a standard object-oriented programming language without explicit simulation support. ERIC was a late addition to our study, at the request of its developers at Rome Laboratories. We did not analyze the features provided by it, and two of the benchmarks were not completed in it.

In this work, we focus exclusively on languages and tools that provide at least a minimum amount of explicit support for object orientation. The advantages of object orientation as a structuring methodology are, by now, well-known. However, the emphasis in object-oriented systems on making decisions at run time can result in significant performance overhead. Our survey attempts to evaluate performance of the systems in quantitative terms, and to qualitatively assess the success of the merging of object orientation and simulation paradigms in the systems.

We designed several concise benchmarks to compare performance of particular features, and one larger simulation to compare general performance. In developing the benchmark code, we noted significant differences between tools in three areas: the modeling approach encouraged or required by the tools, the degree to which the features of object-oriented programming are supported, and the interaction of the modeling considerations with the object-oriented features.

The simulation tools that we consider provide and support a variety of simulation constructs. ERIC provides support for only event-driven simulation. Two of our benchmarks require a notion of interrupt, which is best understood in the process-driven approach to simulation, and these benchmarks were not coded in ERIC. For Smalltalk-80, we used a simulation executive based on the one given in the Smalltalk "blue book" [Goldberg 83]. ParcPlace Systems, the developer of Smalltalk-80, provides no explicit support for simulation. We also examined the option of starting from an object-oriented programming language, and building a simulation executive of our own. One of us (Leivent) designed and built the MOOSE C++ library for simulation. The design and implementation of MOOSE were undertaken when preliminary benchmarks on some of the commercial tools suggested that they could not handle large simulation applications efficiently. The C++ language was chosen for MOOSE because it provides a rich set of object-oriented constructs without incurring excessive performance penalties. The MOOSE system provides the same basic simulation primitives as the commercial tools, and was designed to have a programming interface somewhat resembling that of MODSIM II.

The remainder of this paper is organized into five sections and several appendices. Section 2 provides a discussion of issues in simulation methodology and object-oriented programming. Section 3 contains short descriptions of the tools that we used. Section 4 describes the benchmark simulations. Section 5 discusses the results of the performance testing. Section 6 provides a summary and concluding remarks. Finally, for each tool, the benchmark source code has been included in an appendix.

SECTION 2

DESIGN ISSUES

Many of the differences between object-oriented simulation systems can be grouped into three areas: the modeling approach encouraged by or necessary to use the tools, the degree to which the features of object-oriented programming are supported, and the interaction of the modeling considerations with the object-oriented features. The following section delineates these issues in order to form a framework for the discussion of tool features to be found in section 3.

APPROACHES TO MODELING

Event Selection Strategy

Simulation languages have been characterized successfully by event selection strategy as: event scheduling, activity scanning, and process interaction [Kiviat 71, Fishman 73]. Figure 5 in [Hooper 86] characterizes these strategies in detail. Key consequences of the event selection strategy include:

- What are the components that the modeler develops?
- How is the state of the simulation components expressed?
- How do the components interact with each other and with the system, i.e., what is the world view of the component?

Earlier simulation languages developed in the United States implemented the event scheduling strategy, while the activity scanning strategy gained some popularity in Europe. Many later simulation languages (and later versions of older simulation languages) have adopted the process interaction strategy. It is widely recognized that the process interaction strategy results in a model representation that is "closer to the problem" and thus results in easier and more straightforward model development.

This correspondence of real-world problem to expression in code has been recognized as one of the advantages of Object-Oriented Programming (OOP). This is not a coincidence since much of the early motivation for OOP can be attributed to Simula-67 (which evolved from the simulation language Simula I) and the evolution of other simulation languages. Thus, it should not be surprising that all of the object-oriented simulation tools examined in this paper incorporate the process interaction strategy.

World Views

Certain simulations, such as the bank example described later, differ greatly in design when written with the different tools, despite the fact that all of the tools share the process interaction strategy. These differences arise because of biases toward particular design decompositions (or world-views) that are either supported or required by the tools. In particular, the identification during the design decomposition of *active* versus *passive* components varies according to the tool used.

Active components are defined as those capable of initiating activities, while passive components are incapable of initiating activities. Each active component usually has its own independent time-line, while passive components usually pass time only in a synchrony with an active component, e.g., a passive resource that is held by an active job. Active components usually have more complicated behaviors, so code tends to be concentrated in their representation.

Bezivin [87] has defined two extremes of object-oriented decomposition. In one extreme, which we will call *entity-oriented*, the active components interact by exchanging messages, which are the passive components. In the *thread-oriented* model, the active components may send messages to the passive components, but not vice-versa. The passive components mediate the communication between active components, which never communicate directly. As an example, consider modeling a road traffic simulation problem. The entity-oriented approach would model the crossroads as objects that send the vehicles as messages between them. In the thread-oriented approach, the vehicles are the active clients of the crossroads and make decisions to go from one crossroad to another.

Development of an entity-oriented model generally begins with a decomposition of the system being modeled into *places* at which processing is done. These places, also called entities or nodes, become the active components of the simulation. The *events* or *transactions* that represent the interactions of the entities (the work flowing through them) become the passive components of the model. Modeling then becomes primarily a process of describing precisely and correctly the behavior of each of the entities in response to all of the possible sequences of stimuli, although some consideration must also be given to the information carried by the events.

In contrast, the development of a thread-oriented model concentrates on the flow of processing through the system being modeled. Consequently, active components are sometimes called mobile components, while passive components are called stationary components since they are used to represent fixed services provided to active components. Once the major processing threads are identified, the modeling process is primarily one of specifying the processing steps taken by each of the threads, although the resources or services acquired by the active components must also be described.

The distinction between these two approaches may not yet seem significant to some readers. Indeed, the authors do not know of cases of systems that could not be represented in either of these world-views. However, there certainly are systems for which one of the approaches is better suited than the other. The complexity of behavior of the components of the systems and the bias of the approaches to providing more powerful constructs for the active components than for passive components determines the suitability of one approach over the other. Consider two seemingly similar problems: the road traffic problem introduced above and a train traffic problem. In the road traffic problem, the driver of the vehicles makes the decisions as to which route to take, and the designations of vehicles as the active components and the crossroads as the passive components are most natural. However, in the train traffic problem, the state of the switches at the intersections determines the route of the train. So, it seems most natural to specify the switches with active components and represent the trains with passive components.

The decomposition strategies discussed above are the extremes. In most real-world systems, the distinction of active and passive components is not as straightforward. In the road traffic problem above, the presence of traffic lights would certainly affect the outcome of the simulation. However, the state of the light cannot be said to be an inherent part of the behavior of the vehicle, indicating that the crossroad should be able to initiate activities such as the release of vehicles when a light turns green, thus indicating that the cross-roads should also be an active component. But, since active components cannot interact directly with each other, alternate methods are usually implemented. Also consider the train traffic problem. To say that the trains are completely passive ignores the fact that they may break down or otherwise deviate from their schedules.

There are also systems for which identification of the more active components is not clear-cut. Delcambre [90] considers an apparel manufacturing operation consisting of a number of workstations containing specialized equipment, a number of employees that operate the equipment at the workstations and may be qualified to operate only certain equipment, and job orders that specify the apparel to be manufactured. The job orders contain the information that is used to determine the processing steps involved in completing the order. Each step in the processing requires a workstation with the appropriate equipment and an operator skilled in the operation of the equipment. First, consider a thread-oriented model of this problem. The job orders, or perhaps more precisely the jobs themselves, can be the active components since they specify the threads of processing through the system. The workstations are obviously passive components that perform services for the active components. The workers, however, cannot be easily categorized. To the job, they are resources that must be acquired, and so would seem to be passive components. However, they are active with respect to staffing the workstations since they are constrained by their qualifications, implement the shop manager's scheduling policy or determine their own work preferences, take coffee and rest room breaks, and generally behave in ways that managers abhor. Forcing the developer to model the workers as strictly active or strictly passive forces an unnatural structure on the simulation and may result in ungainly artifacts. Attempting to apply an entirely entity-oriented decomposition results in the same dilemma, even though the assignment of the other components would be the reverse

of that in the thread-oriented decomposition: workstations would be entities sending job-orders as messages to each other after completing their portion of the job, and workers would still be problematic.

The consequences of a bias toward a particular world view are significant to the implementer. As mentioned, more code is generally written for active components, even more than is a natural consequence of the more complex behavior of the more active components. This code may also have to include artifacts of simulation such as event scheduling, random distribution generation, and data collection.

While the bias of a tool toward a particular decomposition strategy may determine the suitability of the tool for a particular problem, the flexibility of a tool in accommodating a number of decomposition strategies will determine its usefulness for a broad range of problems. The tools evaluated differ both in their bias in the world-view that should be used in models developed and in their flexibility in supporting the different world-views. MODSIM II, MOOSE, and Smalltalk are thread-oriented, while *Sim++* and SES/*workbench* are entity-oriented. Each has features that support models of the other world view to varying extents. MODSIM II provides trigger objects to synchronize two active components. *Sim++* events may contain C++ objects with their own methods. SES/*workbench* transactions contain an identifier that other transactions may use to specify synchronization. Smalltalk is unique in that it is completely object-oriented and open. It can be modified to be entity-oriented. Generally, tools that are less biased to one extreme are also more flexible.

Other consequences of a bias in world view include limitations on support of object-oriented development and the ability to support parallel simulation execution. A strong distinction between passive and active components may weaken the passive component's role as a "first-class" object in the development process. It may preclude the ability to derive passive objects by inheritance or otherwise customize the behavior of the passive components. Most attempts at parallel simulation have adopted the entity-oriented world view, since it seems to result in a relatively small number of components of sufficiently large granularity to overwhelm communication and synchronization overheads.

The event selection strategy and the world view of a tool combine to determine the overall modeling approach of the tool. As mentioned, all of the tools we investigated incorporate the process interaction strategy. In all of these tools, the active components are processes. The simulation primitives available to these processes include many that resemble primitives used in parallel programming, such as synchronization, interrupt, and delay constructs.

SUPPORT FOR OBJECT ORIENTED DEVELOPMENT

The benefits of object-oriented development have been extensively debated in the literature. The application of object-oriented development to simulation has not been as extensively examined, and has generally focussed on the productivity benefits in development [Eldridge 90]. The benefits of object-orientation in the modeling process have recently been examined [Delcambre 90].

The tools evaluated here vary widely in their support for object-oriented development. They also differ greatly in the integration of "programming objects" with the "simulation processes."

Integration of Objects and Simulation Time

Object-oriented development results in a set of partitions of a program's data space and execution trace that are called objects. The benefit of the object-oriented paradigm over other module-based paradigms is that the resulting modules include both data and the code to manipulate that data together to form abstract data types. One premise of object-oriented development is that the resulting objects encapsulating the abstract data types are safer and less likely to be misused, since the modules are more cohesive and their intent is captured abstractly. Also, the objects are easy to modify as the requirements of the program evolve or become better refined. The process of design in object-oriented development has as a goal the delineation of the objects that will be implemented in order to fulfill the requirements of the program. Object-oriented design often includes a modeling process, where the objects in the real-world problem are identified and abstracted for representation in code.

Similarly, modeling in the process interaction approach to simulation also includes the identification of the real-world components of the system being modeled. The identified components encapsulate the processes, sequences of activities necessary to perform the work of the system being modeled. Each process contains pieces of the execution trace that represents the flow of simulation time. Execution will continue within one process as long as consecutive steps in the processing of the system can occur and jump to other processes when the next processing step cannot occur in the current process.

While the similarities between the two design processes are obvious, the differences can make simulation development more difficult. In general, one cannot simply follow one of the popular object-oriented design techniques and then add the simulation considerations later. One reason for this is that present object-oriented methods are based on a static semantics, i.e., the passage of time is only a side-effect of the execution of the functions and procedures that act on the objects. Coordination between objects results only from a need of one object to invoke the processing encapsulated within another. Newer methodologies are starting to incorporate concurrent semantics, i.e., the notion that there may be several concurrent threads of execution that must be synchronized at certain points. This is a closer match to the approach of process-interaction simulation design, in which the coordination of processes is required only when the processes must synchronize in simulated time.

This interaction of object-oriented design and process-oriented design results in restrictions on where object boundaries are drawn, on which modules contain the time lines, and how each object can elapse simulation time. Different consequences were found in each of the tools examined.

Where are the Boundaries?

The boundaries between objects are shaped not only by the object-oriented decomposition process but also by where advances in simulation time occur. As an example, consider a model in which a job must acquire a resource. Since it is presumed that the resource may be acquired by a number of jobs, the resource has its own flow through simulation time, i.e., it has its own time-line (at least conceptually; it may be modeled as being atomically attached to the time-line of its acquirer). Thus, the procedure (or method) that acquires the resource coordinates across time-lines and the mechanics of the discrete-event semantics must be invoked. The visibility of these mechanisms differs between the tools examined. At one extreme, MODSIM II simply requires that the method for acquiring a resource be designated as one that may have simulation time side effects. At the other extreme, *Sim++* requires the explicit passing of an event from one process, the acquirer, to another, the resource. In the former case, the method for acquiring the resource appears completely as part of the resource. In the later case, portions of the method appear in two entities. Thus, the boundaries between objects are drawn differently in the two tools.

Where are the Time Lines?

The interaction of the simulated time lines with the objects, i.e., the granularity of the domains in which simulation time is constant, also vary considerably. In most tools, certain objects are designated to be the simulation-relevant entities, i.e., the objects that define the simulation time at which its own methods and the methods of subsidiary objects execute. However, one tool (MODSIM II) allows each method to have its own time, even within objects. Thus, an object could be executing each of its methods concurrently in simulation time.

What Entities Can Elapse Simulation Time?

A related consideration, at least in the tools in which there are simulation-relevant entities as opposed to other entities, is whether the non-relevant entities can cause simulation time to pass, and if so in what domain. The solutions vary from not allowing objects that are not simulation entities to pass simulation time (SES/*workbench*), to allowing non-relevant objects to pass time for the simulation entity which directly or indirectly invokes the object's method (*Sim++*), to allowing each method to affect only its own execution trace (MODSIM II).

Inheritance

Inheritance is the most popular of the mechanisms in object-oriented programming (as distinguished from merely object-oriented design) that allow related sets of objects to share common implementations of abstract data and methods and to customize these to produce slightly different behavior. The ideas of inheritance are borrowed from the classification methods of biology and other natural sciences. For example, the attributes of an animal include producing progeny whereas the attributes of a mammal are generally specialized to include live birth from the mother and require nourishment of the young with secreted milk. The software program for a hospital may include a class of objects representing rooms, that have attributes such as length and width and methods such as assignment of a patient. A specialized room would directly inherit these attributes and methods unless they were over-ridden. For example, an isolation room would have the inherited attributes of length and width, but would over-ride the assignment method so that only patients that have been determined to be dangerously contagious would be assigned to them. The mechanics of specifying the similarities of the specialized room to any other room and of differentiating the room from others is provided by the inheritance mechanism of the software development system.

The support for inheritance, or other sharing mechanism, varies within the tools. Some do not support any sharing mechanism other than the creation of a number of instances of an object, whereas others provide full support. In some of the tools, inheritance is complicated by the restriction on drawing boundaries between objects. If we return to the resource example above, the partition of the "acquire" method across two simulation objects complicates the derivation of a subclass of the resource class.

Multiple inheritance is a means of specifying derivation of a class of objects from two or more parent classes. Other than the mechanics of specifying this inheritance, the issues associated with multiple inheritance include resolution of conflicts when two parents provide methods or attributes of the same name. Some of the tools examined do not support multiple inheritance, while those that do differ in the method of conflict resolution.

Strong Types and Object-Orientation

The simulation tools we have investigated fall into four categories with respect to their object-oriented behavior. Smalltalk is exclusively object-oriented (everything is an object, every piece of code is a method) and has no typing mechanism for variables. The C++ tools (MOOSE and *Sim++*) are object-oriented, but not exclusively (there exist data representations that are not objects; there are pieces of code that are not methods) and have a strong typing mechanism for variables. MODSIM II exhibits a subset of the object-oriented functionality of C++. SES/*workbench* is actually object-based, since it lacks an inheritance mechanism.

The Smalltalk style of object-oriented programming is perhaps the oldest and most well known. Smalltalk's lack of any typing mechanism for variables is most beneficial in the areas of rapid prototyping and iterative refinement of software. Also, there is little argument about the elegance of the non-typed object-oriented style: Smalltalk's semantics are far easier to

understand and work with than any of the other tools studied here. However, a strong typing mechanism is missed in the areas of program readability and understandability and, as our benchmarks show quite clearly, performance.

The question is, when a strong typing mechanism is present, is the loss of rapid programmability and refinability worth the gain in performance. As is demonstrated by our benchmarks, the performance benefit may be so overwhelming that all other motivations can be suppressed. This is especially true for large simulations. For smaller simulations, and especially for simulation prototyping, the performance benefits may not be so overwhelming. The degree of integration of strong typing into the object-oriented constructs in the C++ and MODSIM II models may be part of the decision of which tool to choose.

The lack of any typing semantics in Smalltalk means that a Smalltalk variable can refer to any Smalltalk object. Furthermore, messages are resolved to methods based solely on the type of the destination object, and this resolution is always done at run time (commonly known as late binding). This variant of object-oriented semantics makes the implementation of generic structures, such as collection classes, very easy and natural.

The semantics of the combination of strong typing and object orientedness in MODSIM II basically involves the limitation of the values of variables to objects having a specific common ancestor class. A variable of object type X can refer to object Y if and only if the object type of object Y is a descendent of object type X. There is a single type, called ANYOBJ, to which a variable can be typed so as to be allowed to refer to any object. Assignments between variables of type ANYOBJ and variables of any other type are permitted. However, references to an object's instance variables and methods cannot be made through a variable of type ANYOBJ.

The purpose of MODSIM II's strong types seems to be related to the software engineering goal of program clarity. There is agreement among the authors that MODSIM II does accomplish this goal very well relative to the other tools investigated here. However, the issues of performance and ease of programming are not similarly addressed. MODSIM II methods are all late binding despite the presence of strong typing, so messages are less efficient than function calls. Also, overriding methods in subclasses is hindered by the requirement that the signatures of the overriding and overridden methods be identical. This particular rule can complicate the process of extending the functionality of a class through the formation of subclasses. One immediate impact is that the object initialization method ObjInit cannot have any arguments, making it much less useful than object constructors in Smalltalk and C++.

C++, possibility because of its kinship with C, focuses primarily on how strong types can increase the performance of object-oriented programs. Unlike both MODSIM II and Smalltalk, most methods in C++ are early binding, allowing the compiler to translate message sends directly into function calls without any additional run-time search. Late binding can be achieved through the use of virtual methods which have a small associated performance penalty.

Unlike MODSIM II, C++ possesses overloading semantics, which allows multiple methods with the same name and different argument signatures to exist without difficulty. This permits developers of subclasses to extend the functionality of superclasses by adding and/or changing arguments when overriding methods. A further advantage of overloading is the ability to overload most of the operators in C++, including arithmetic and logical operators, comparison operators, the assignment operator, dereference operators, and the function application operator.

Assigning between variables of different class types in C++ can be tricky. The actual rule for such assignments is something like: assignment between variables of different class types is permitted directly if the type of the source variable (or expression) is a descendent of the type of the destination variable; assignment in the opposite direction from ancestor to descendent is possible using casting, but it is only safe if the destination variable type is a leftmost ancestor (either the first listed parent class, or the first listed parent class of the first listed parent class, etc.) of the object's class, or if the destination variable type is a virtual ancestor of the object's class. This rule can complicate the task of writing fully reusable methods, especially for generic structures such as collections. Other rules involving class typed arguments to functions and methods, and how overloaded calls are resolved, are also complex. In fact, one rule in C++ that allows a derived class reference (a generalized variable of a descendent class type) to be implicitly converted to a public base class reference (a generalized variable of an ancestor class type) allows unsafe assignments to be performed without so much as a warning.

Dynamic Creation of Simulation Objects

Some problems are best modeled with models that require the creation of active simulation components. Consider a model of a typical multi-user computer system where programs run within operating system processes on processors. Since processes are created dynamically by the operating system in response to users or user programs, they cannot be statically created at initialization and yet are complex enough that they should have their own associated timeline. Thus, it might be important to the modeler to have the capability to dynamically create simulation objects.

Two of the tools (*Sim++* and SES/*workbench*) have re[illegible]ions that prohibit the creation of simulation objects after either development or after an i[illegible]l phase. Not coincidentally, these tools are also the ones that are most entity-oriented.

SIMULATION CONSTRUCTS

Time Control

It is a tautology to say that time control mechanisms are required for simulation. However, there have been a wide range of time control mechanisms implemented in different simulation languages. Much of the difference in these mechanisms is directly attributable to differences in either the event selection strategy or the world-view supported by the modeling tool. These considerations have already been discussed.

All of the tools examined provide time control mechanisms that are more than adequate for any problem which we were able to conceive.

Time control mechanisms differ markedly in their visibility, however. In some tools the passing of events is explicit, while others hide some events, such as the completion of a hold, and in others events are never visible.

Preemption

One important time-control mechanism that caused some trouble in earlier simulation languages is the ability to preempt or interrupt a process after it has started. This capability has a broad range of applications.

The support for preemption, like other time control issues, is tied up with the other modeling concerns. The tools have widely different implementation mechanisms.

Pre-Defined Classes

Pre-defined classes can be used to represent parts of the modeled system that conform to the behavior defined by the class. These pre-defined classes, when they can be used, cut development time and size. If the extensibility of the tool is restricted, as discussed below, the pre-defined classes may define the range of applicability of the tool.

All tools that favor the thread-oriented decomposition provide a resource class. A resource is a depository of a number of tokens that can be acquired, held, and given back either singly or multiply. An attempt to acquire one or more tokens when the requested number are not available results in the blocking (in simulation time) of the acquirer.

The tools differ in number and types of pre-defined classes. This is discussed further in the next section, where we cover the tools individually.

Probability Distributions

Random number generation is an important part of most simulations. The tools examined vary only slightly in the number and types of random distributions provided. We did not undertake any evaluation of the quality of the generators. During our benchmarking, we did experience a problem with random number generation in Smalltalk-80. A distribution which should have returned only positive numbers returned zero on occasion, presumably due to round-off error.

Data Collection

Data collection support includes support for accumulation of statistical data, statistical analysis, and I/O operations to allow archives. All of the commercial tools examined provide very similar capabilities. Data collection in MOOSE is not implemented.

EXTENSIBILITY

The history of simulation tools has supported two trends: the extension of an existing general purpose language to include simulation support, or the creation of a special-purpose simulation language. The first presumedly provides greater extensibility, while the latter presumedly provides greater integration and ease of use.

Different problems require different degrees of extensibility. Of the tools examined, three (Smalltalk, Sim++, MOOSE) are extensions of existing general-purpose languages, while the others (MODSIM II, SES/*workbench*) are simulation-specific developments. Of these, one is claiming to be robust enough for general purpose use, while the other is extensible through its own language or through its translation to C.

GRAPHICAL INTERFACES AND ANIMATION

Graphical interfaces are being used in simulation in both the development process and in the display of results. SES/*workbench* provides a graphical interface for the development of models. Instances of pre-defined object types are selected from a palette, positioned within a window, and connected using Macintosh-like point-and-click methods. Pop-up boxes are provided for forms that further parameterize the behavior of the model components. The latest release of SES/*workbench* also provides animation capability.

MODSIM II provides a library of graphical objects which can be used to animate the results of the simulation or to present the results in graphs or other presentation graphics.

SECTION 3

TOOLS

Our selection of tools was biased by what was already available at our corporation and what we could acquire for reasonable cost. There are many interesting simulation systems that we did not consider. For example, SimKit (with KEE) from IntelliCorp provides a wide range of simulation and expert system capability. Also, other object-oriented programming languages, such as Eiffel and Simula, have not been considered. LISP, as the base language of ERIC, has been involved in our study, but only to a limited extent.

In the subsections below, we provide a summary of the capabilities of the commercial products that we did consider, and then a description of MOOSE.

MODSIM II

MODSIM II is a "general purpose, modular, block-structured high-level programming language which provides direct support for object-oriented programming and discrete-event simulation" [Belanger 90a, 90b]. CACI Products Company markets MODSIM II as the commercial version of ModSim, which was created on a US Army contract. Modula-2 was the base language used in the creation of ModSim.

Simulation in MODSIM II is thread-oriented. Threads are created by specially designated methods, called tell methods. A tell method programs the events that will occur in the thread. Tell methods are asynchronous and cannot return values; when one is called, a new thread is created and the calling unit continues its execution. Tell methods are also reentrant, meaning that a new thread can be started while other copies are running. An ask method is the more traditional method call, in that the calling unit waits until the ask method completes. One of the limitations of MODSIM II is that simulation time can be elapsed only directly inside tell methods. Thus, if a tell method calls an ask method, that ask method cannot directly execute a wait statement.

The object-oriented features of MODSIM II are sometimes restricted to agree with the type structure. In particular, a method can only be overridden by another method taking precisely the same arguments. Multiple inheritance is supported, and ambiguous references are flagged as errors.

Code in MODSIM II is written in separate main, definition and implementation modules. The system comes with a smart compilation tool, mscomp, that can build a complete simulation from a main module, recompiling and linking the appropriate submodules. The compiler for MODSIM II generates C as output.

One of the unique capabilities of MODSIM II is that it supports an interface to CACI's graphics package, SIMGRAPHICS. CACI claims that animated simulation demonstrations and interactive I/O are facilitated by SIMGRAPHICS, but we did not test these features.

SES/WORKBENCH

Scientific and Engineering Software, Inc. (SES), introduced SES/*workbench* in March of 1989. Workbench is based heavily on queuing theory, having evolved from the earlier PAWS/GPSM (Performance Analyst's Workbench System / Graphical Programming of Simulation Models). Our tests were performed using release 1.11 of Workbench, which was the most recent version until February 1991, when Release 2.0 became available. Release 2.0 reportedly contains animation capability, which is completely missing from Release 1.

A unique feature of SES/*workbench* is the graphical front end, SES/*design*, which allows specification of a simulation without programming. In SES/*design*, a simulation is specified as a hierarchy of directed graphs. Simulation threads are called transactions in Workbench. Transactions flow along arcs in the directed graph. Nodes in the graphs can manage transactions, e.g., source nodes, which create transactions, or manage resources, e.g., allocate nodes, where a transaction queues for a resource. A small set of standard predefined nodes is supplied, together with a user node that must be coded by the user in C. The events in a transaction are not directly programmed, but arise as the transaction traverses the graph. For this reason, we view Workbench as entity-oriented. Transactions, however, do play an important role in Workbench. Mechanisms exist for naming transactions and interrupting them at arbitrary points in their execution.

The graphs created by SES/*design* are stored as ASCII files. These files are compiled by Workbench into a simulation language, SES/*sim*. This language is a superset of C, containing extensions that were influenced by PAWS and by C++. Users can program directly in the simulation language, if they desire. For our benchmarks, we used the graphical interface. Our main complaint is the difficulty of debugging. Errors in the graph file are usually not discovered until the simulation language is compiled into C. The generated error messages refer to line numbers in the machine-generated simulation language file. This leaves the user with the problem of trying to trace an error back to an arc or node in the graphical input. Some improvements to debugging are claimed by SES for Release 2.0.

Object-orientation is not an emphasized part of Workbench. The SES/*sim* language does contain constructs for specifying classes and creating instances, similar to C++. The SES/*sim* manual lists only very basic facilities for object-orientation. In particular, there appears to be no provision for declaring base classes or member functions to be public or private, no friendship mechanism, no operator overloading — in short, most of the more elaborate constructs of C++ are not present. The object-oriented features that do exist are more likely to be used by the SES tool than by the simulation designer.

Sim++

Sim++ is a C++ library of simulation constructs produced by Jade Simulations International Corporation of Calgary, Canada. The unique feature of *Sim++* is support for parallel execution using the TimeWarp Distributed run-time system. The later versions of Sim++ require Release 2.0 of ATT C++, which the user licenses separately. Jade recommends 8 nodes as the minimum reasonable parallel configuration. Networks of Sun -3 or -4 workstations, the BBN Butterfly, and the Meiko Computing Surface transputer array are the supported hardware. The Distributed run-time environment provides deterministic execution despite being distributed. A number of tools are provided to increase execution speed-up. *Sim++* also provides an Optimized Sequential run-time executive for developing, debugging, and executing simulations on a single machine. The optimization removes most of the execution overhead associated with parallel execution.

The results in the following sections were obtained by using Release 3.0 of *Sim++* on a single workstation using the Optimized Sequential run-time system. While it might be expected that the emphasis on performance of the sequential executive is not as great as that for the distributed executive, and that the benchmark results for *Sim++* might suffer as a result, we did not use the distributed executive for several reasons. First, we did not have it. Second, the single-feature benchmarks would not have benefitted from parallel execution. Finally, the characterization of the performance of parallel systems is more complicated in general and was felt to be beyond the resources available.

The *Sim++* simulation approach is entity-oriented. A static set of simulation entities is created for each simulation run from sub-classes of the *Sim++*–provided sim_entity class. These sim_entity sub-classes define the behavior of the entities in response to receiving (or failing to receive) events. Events passed between entities are derived from the sim_event class that includes an integer field for event typing and a pointer to allow inclusion of a body containing state information in the event. While an event body may be any C++ object, there is no enforcement of consistency between the integer event type and the supplied event body. This consideration and the lack of a mechanism to directly tie event types to entity methods tends to limit the usefulness of inheritance to defining components of entities and events rather than whole entities or events.

Preemption is supported by a Hold_For construct that is interrupted by either any event or an event that passes a selection criterion. Selection criteria include any combination of event originator, event type, or contents. While this construct may not be as readable as the interrupt mechanism in MODSIM II, it may be more flexible.

While there is no pre-built support for resources, resource and consumer classes were built fairly simply for the fourth single-feature benchmark described in the following section. These classes used the Hold_For construct. Events requesting a resource were deferred while a resource was held by another requestor. After release, the next requesting event is selected from the system-managed deferred event queue.

While no explicit support for graphical input or animation was provided, the multiple inheritance feature of the C++ base of *Sim++* allows easy extension by integration with other libraries.

SMALLTALK-80

Smalltalk is a general-purpose, object-oriented programming language. For our tests, we used Smalltalk-80, a product of ParcPlace Systems. We had access to Release 2.5 on Macintosh hardware, and Release 4 (the successor to 2.5) on Sun workstations. A collection of simulation constructs for Smalltalk is described in the Smalltalk "blue book" [Goldberg 83] and is implemented in Smalltalk-80. The constructs encourage the thread-oriented approach, but the entity-oriented approach can also be used. There is a useful and general approach to passive and active resources. No provision is made for interrupts, but this was easily fixed. One of the main advantages of Smalltalk is the open nature of the system, with full source code visible to the user. For simulation, the event queue mechanisms can be examined and changed, if desired. In the browser tool, we were able to add interrupt mechanisms to the simulation constructs. The new constructs merged seamlessly into the existing ones.

There are three possible problem areas in Smalltalk. The first, and most important, is the performance problem. As a rough rule of thumb for general computing, Smalltalk is about one order of magnitude slower than optimized C [Chambers 89]. Doyle's data confirms this rule for a simulation benchmark [Doyle 90], and our timing studies show similar results. For simulations where performance is not a critical factor, Smalltalk may be a very good choice. The second potential problem is the lack of multiple inheritance. There was at one time an experimental implementation of multiple inheritance in Smalltalk-80 [Borning xx], but it was eliminated after version 2.3. Currently, only single inheritance is supported in Smalltalk-80. The third potential problem is the Smalltalk learning curve. The programming language and environment for Smalltalk-80 form a uniquely powerful system. The time required to become proficient in Smalltalk-80 is undoubtedly longer than that for MODSIM II or SES/*workbench*. The investment in learning time pays off in increased capability.

A PROTOTYPE C++ SIMULATION LIBRARY -- MOOSE

MOOSE is a C++ implementation of the process model of discrete simulation. This model is most similar to the MODSIM II model, where each TELL method execution is a process. However, unlike MODSIM II, simulation processes are first class objects. Like MODSIM II, MOOSE supports dynamic creation of processes.

The programmer interface to MOOSE was designed to be similar in nature to that of MODSIM II because of the authors' familiarity with that tool, and because of MODSIM II's ease of programmability.

The majority of functionality within MOOSE is provided by class Process. The simulation programmer is expected to provide subclasses of class Process, each with its own definition for the start() virtual function member, and its own set of constructors. The arguments to a process are provided through the constructors, and are stored within data members of the process object. The start() member function is called by the process scheduler to initiate the process. MOOSE provides several scheduling primitives that can be called from anywhere within a process' execution.

Processes in MOOSE can be created dynamically, and are expected to have varying lifetimes. The memory consumed by a MOOSE process is reclaimed when the process terminates. The MOOSE programmer is protected against dangling references to processes that have been garbage collected after termination by the use of a safe referencing scheme implemented by the process identifier (PID) class.

MOOSE is implemented using only portable C++ functionality. The process class is implemented using the setjmp and longjmp functions (from the standard C include file setjmp.h) to create coroutines on the execution stack. Such an implementation of processes can run in any C++ environment. However, the use of virtual memory machines is strongly recommended for simulations of any significant size because the setjmp/longjmp coroutining technique uses large amounts of address space (the system allocates 4K bytes by default for each process' stack), even though the amount of virtual memory actually used may be low (many processes use only a small portion of their stack).

The MOOSE event list used for scheduling processes has a tightly coded heapsort-based priority queue implementation. This implementation was found to be slightly faster on both Sun-3s and Sun-4s than several alternatives in the O(NlogN) category, such as splay trees and leftist trees. The heapsort algorithm is array-based, and requires that the heap array be allocated statically. However, the heap array is reallocated (using the realloc function) as needed. The additional complexity of the reallocation of the heap array, including the check for overflow prior to every insertion, did not prevent the heapsort-based implementation from running faster than the others tested (see the results of the Test 1 benchmark).

The process scheduling primitives in MOOSE, including process waiting and interrupts, together with the fact that MOOSE processes are directly accessible, have been shown to be sufficient for the implementation of many diverse simulation constructs, including resources and triggers.

ERIC

ERIC is an object-oriented simulation tool designed and developed at Rome Labs [Hilton 1990]. Initially, ERIC stood for Enhanced ROSS in Common LISP, but as ERIC was developed, it diverged from ROSS (a simulation tool from RAND Corporation) and the name is no longer considered an acronym. Compared to the other tools that we considered, ERIC is unique in that it is event-driven. We completed our first four benchmarks for ERIC, but the

fifth benchmark and the bank simulation require a notion of interrupt. We could find no simple way to model interrupts in ERIC, due to its event-driven nature, and hence we did not complete the last two benchmarks for this system. For our tests, we used a version of ERIC in Allegro Common Lisp with the Common Lisp Object System (CLOS).

SECTION 4

BENCHMARKS

The design of performance benchmarks for object-oriented systems seems to be an uncharted area. For simulation systems, Doyle [Doyle 90] studied several tools using a single benchmark. We have chosen five feature-specific benchmarks, and a single general purpose benchmark. The benchmarks were implemented and timed in each of the tested tools.

SINGLE FEATURE BENCHMARKS

These are small benchmarks designed to test single features of the simulation tool. Each is parameterized by a single integer input usually representing the number of threads generated (test 5 is the exception; the integer parameter in that test represents the number of interrupts generated). The results are graphed and discussed in the next section. Abstractly, a thread is a sequence of causally related events operating on the same state information. Exactly how a "thread" is implemented is different for the different simulation tools. In the transaction-based simulation models, a thread corresponds to a transaction. In the process-based simulation models, a thread corresponds to a process. Within a thread, at most one event can occur at any simulation time.

Test 1 - Sorting Threads

Initially, N threads are created. Each thread is given a starting simulation time chosen from a uniformly distributed random variable. The threads simply terminate as soon as they are started. The system must sort and execute the threads. Asymptotic performance on this test ranges from nearly linear for MOOSE to quadratic on some of the commercial tools. Also, for *Sim++* and ERIC, two tests were performed to illustrate the difference in performance when each thread is associated with a different object and when all threads are associated with the same object. In both *Sim++* and ERIC, the event list sorting algorithm's asymptotic performance is better for the case when each thread is associated with a different object.

This thread sorting test is expected to predict the relative performance of the simulation tools on simulations containing a large number of threads. Had all of the tools used similar sorting procedures, the test would not be an accurate predictor. However, because of the quadratic behavior of some of the sorting algorithms, the simulation systems with nearly linear behavior (actually, O[NlogN] complexity) are clearly favored for large simulations over those with quadratic behavior.

Test 2 - Thread Creation

This test is designed to compare the overhead involved in creating and manipulating individual threads (all for the same simulation object) without the overhead associated with thread sorting measured in test 1. For this test, a single thread is initially created which spawns a child thread and then terminates. The child thread then spawns a third generation thread and terminates, and so on until N threads have been generated. At any time, there is at most one thread waiting to execute, so the overhead of sorting is not incurred.

Differences between the semantics of threads in the tools compared should be noted when examining the results of this test. Those simulation tools that have process semantics for threads (Smalltalk, MOOSE, MODSIM II) are trading overhead as measured in this test for power within a process, which in most cases would translate to fewer thread creations in these tools for given simulation than for the non-process oriented tools (*Sim++* and SES/*workbench*). MODSIM II is actually somewhat of a hybrid between the process and non-process models, since threads in MODSIM II can only elapse simulation time from within their outermost stack frame.

Test 3 - Synchronous Thread Creation

Modularity issues arise in simulation languages just as they do in standard programming languages. The software engineering ideal for modularity is that there should be a negligible tradeoff of performance for modularity inherent in the language. This, however, is difficult to test, since the notion of modularity is not nearly as formalizable and measurable as is performance. This test is an attempt to show that the implementation of some simulation systems encourages a "demodularization" of code exceeding that normally experienced in standard programming languages.

In standard programming languages, the most common unit of modularization is the function (procedures and methods are here considered synonymous with functions). Small, easy to understand functions that encapsulate simple ideas are preferred for modularity, readability, maintainability, and nearly every other software engineering concern. It is generally accepted that code which localizes concepts is to be preferred. The largest drawback of function modularity is the added overhead of the extra function calls, but this is not a severe performance penalty in most programming languages. For one of the simulation systems investigated here, however, function modularity within threads can impact performance considerably. The problem occurs in MODSIM II, when a single thread must pass through several functions, any of which may or may not elapse simulation time. In all other tools, any function (or function equivalent, such as a subgraph in SES/*workbench* or method in MODSIM II or MOOSE) may elapse simulation time within a thread. It is possible to spawn a new thread that is synchronously tied to its parent (the parent will sleep until the child is done, then the parent will continue), but this is not necessary. In MODSIM II, however, it is

necessary to spawn a synchronous child thread using the WAIT FOR construct to permit the called method to elapse simulation time. This benchmark is designed to demonstrate the impact on performance that this restriction can have.[1]

In test 3, a simulation thread synchronously calls a child function. This behavior is continued to a depth equal to the input parameter. In test 3a, the function calls do not elapse simulation time. In test 3b, each call elapses one unit of simulation time. In both cases, we have coded the test in a manner that would not prohibit the child from elapsing simulation time (which means that a WAIT FOR construct is used in MODSIM II, while direct function calls are used in all other tools).

Test 4 - Resource Queues

Resources are one of the most common constructs found in simulation systems. Resources are generally represented as queues with some standard queuing discipline (usually FIFO) and some number of tokens. Threads can request some of the tokens from a resource. If the resource has sufficient tokens to fulfill the request, the thread is allowed to continue. If the resource has too few tokens to fulfill the request, then the requesting thread is queued and its execution is blocked. Threads then can release tokens back to resources, which may cause the resources to dequeue waiting threads and allow them to continue executing.

Of the simulation systems tested, all but *Sim++* and ERIC contain some built in resource construct with semantics equivalent to that described above. For both *Sim++* and ERIC, resources are implemented as separate simulation objects that use event rescheduling to achieve the desired queuing and waiting semantics.

For this resource test, a resource containing single token is created, and N threads request the resource. Care has been taken to construct this test so that at most one thread at any time is scheduled by thread sorting (as tested in test 1), so that the overhead of the sorting algorithm is not felt. Performance was generally linear here, except for MODSIM II, which exhibited quadratic performance. We have since sent CACI our code for this test. They profiled it to find the performance problem areas and report that Release 1.6 of MODSIM will include improvements.

Test 5 - Interrupts

Interrupts, like resources, are common to most simulation systems. Interrupts present a semantics for the control of threads by other threads. The target thread of an interrupt is always in a wait state, since this the only way that the source thread can have time to initiate the interrupt. The source thread of the interrupt can interrupt the target using some construct that requires a way of denoting the target thread (in MODSIM II, where threads are not

---

[1] Subsequent to our work, CACI has apparently corrected this problem in Release 1.7 with the introduction of WAIT FOR *methods*. When WAIT FOR methods are invoked by a WAIT FOR construct, no context switch occurs.

independently namable entities, a thread is denoted by the host object and the method name - this technique may not always indicate a unique thread). The target thread of the interrupt is scheduled to execute immediately after the interrupt (or, at least before any simulation time elapses), and control within the thread is usually transferred to some interrupt handler.

Of the simulation systems tested, all but *Sim++*, Smalltalk, and ERIC contain a built in interrupt facility. For Smalltalk, an interrupt mechanism was added simply by adding the appropriate methods to some of the built in simulation classes. For *Sim++* and ERIC, an event rescheduling feature was used to obtain the interrupt semantics.

For this test, a source thread and a target thread are created. The source thread will interrupt the target thread N times, each time while the target thread is waiting within a delay construct. The interrupt handler for the target thread simply re-invokes the delay, causing the target thread to wait for the next interrupt. For this test, all tools showed similar performance.

BANK SIMULATION BENCHMARK

The bank simulation is our revision and enlargement of an example supplied with MODSIM II. The purpose of this benchmark is to test many simulation system features together within the context of a "typical" simulation. The simulation consists of "customers" and "VIPs" that enter a simulation of a bank, requiring service. There are a fixed number of identical servers ("tellers") in the system. When a customer enters the simulation, it selects a server with the shortest queue. When a VIP enters, it selects a server at random and attempts to receive immediate service. If the teller chosen by a VIP is serving a non-VIP customer, the customer is interrupted and the VIP is served; if the server is serving another VIP, the requesting VIP simply departs the system in disgust. The servicing of a customer interrupted by a VIP is resumed after the VIP has been serviced.

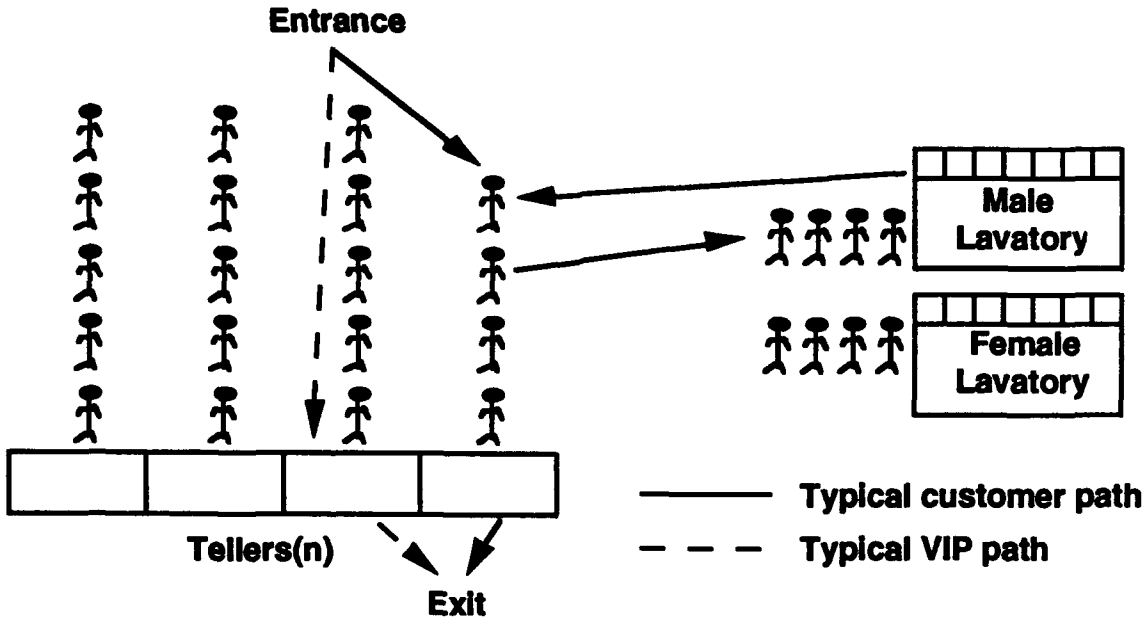


Figure 1. Bank Simulation Sketch

To make the simulation a bit more interesting, while a customer is in a queue, it may "time-out" and be required to visit the lavatory. A lavatory has a number of stalls. Customers visit the lavatory appropriate to their gender and use the first available open stall. After the lavatory visit is complete, the customer again selects a teller with the shortest queue. Furthermore, every customer has a "lavatory line length tolerance" – if the line to the lavatory exceeds this tolerance, the customer will leave the bank and seek "alternate facilities," rather than wait on line. The bank thus has an implicit saturation point, past which a higher rate of arrival of customers will result only in the excess customers leaving the bank. However, the complex interaction of customers, VIPs, and visits to the lavatory makes any analytic determination of the saturation point non-trivial.

The generic benchmark was coded and successfully run on all systems except MODSIM II. Under Release 1.5 of MODSIM II, we experienced run-time errors related to the interrupt constructs. We reported the error and received a beta version of Release 1.6 under which the simulation runs correctly.

SECTION 5

TIMING RESULTS

TEST 1

The timing results for Test 1 are shown in figure 2 below. Since the benchmark tests the sorting algorithm implemented by the run-time environment of the simulation tools, asymptotic behavior of O(n log n) was expected. Surprisingly, most of the tools exhibited quadratic behavior. The exceptions are MOOSE and *Sim++* (when the threads are scheduled for distinct entities). Not as surprisingly, Smalltalk performed significantly worse than any of the other tools.

Note that the "N queues" result for *Sim++* was measured as the difference between two separate tests so that the overhead of creating entities could be removed. The cost of creating entities was significantly more than the cost of creating threads.

The inability to run the SES/*workbench* benchmark for more then 6,000 iterations is unexplained. Our implementation simply never terminated at this input level. The lack of data past 7,000 iterations of the *Sim++* (N queues) implementation reflects the point at which the physical memory of the workstation used was exhausted. After that point the effects of paging could not be separated and the data was discarded.

The authors speculate that the tools are optimized for simulations where the number of events on the queue is not large and, therefore, the constant multiplier may be a greater consideration than the order of the sorting algorithm used. Also, we do not know if using uniformly distributed event times is a suitable approximation to the function of typical simulations, where event times may appear in an almost sorted order.

TEST 2

Figures 3 and 4 show the results of this benchmark. Once again Smalltalk took much more time to perform the same number of iterations as any of the other tools. In fact, the Smalltalk results are removed from figure 4 so that the relative differences of the other tools could be shown clearly. As expected, the cost of creating a thread is roughly linear in the number of threads created.

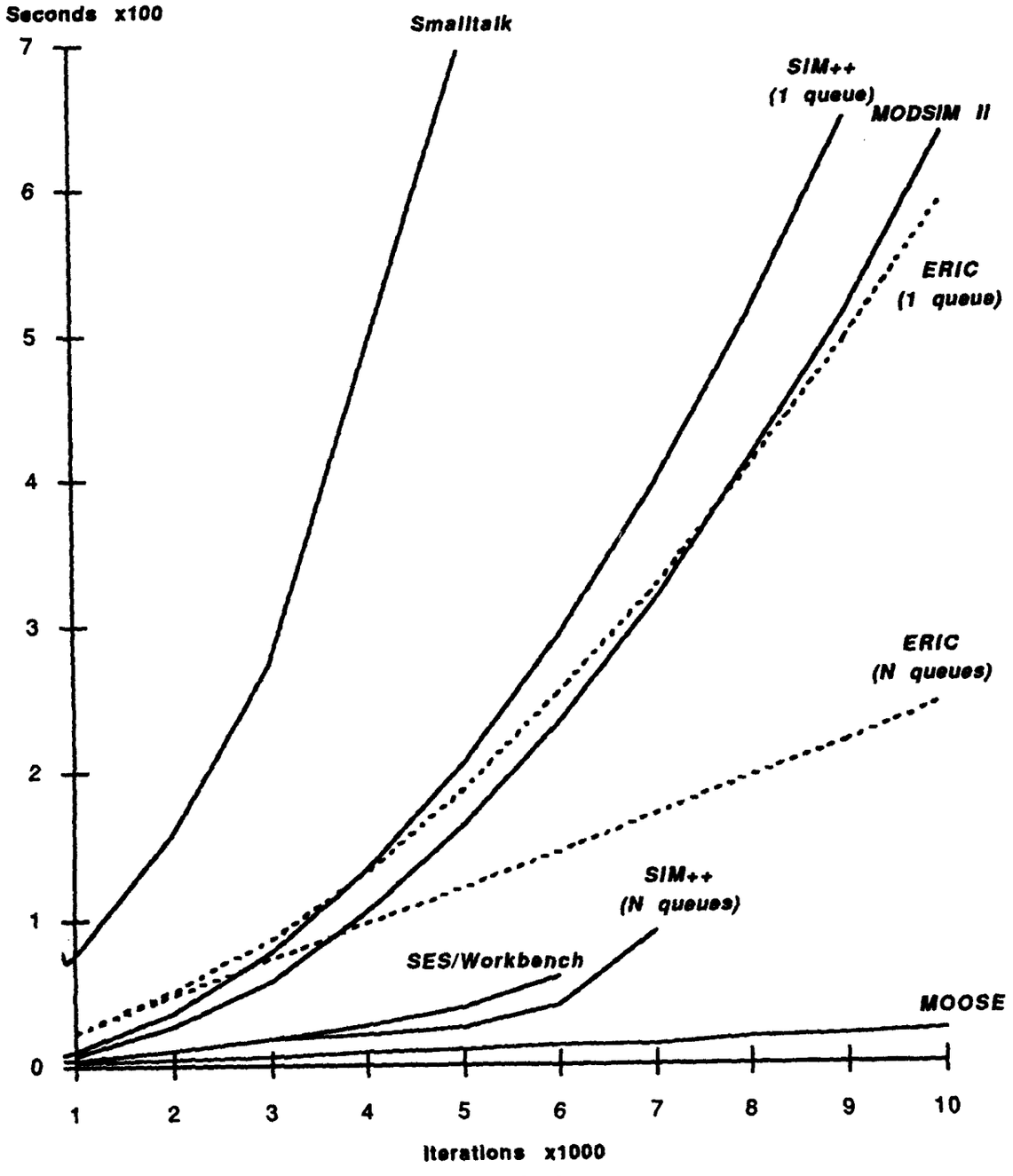


Figure 2. Test 1 Performance

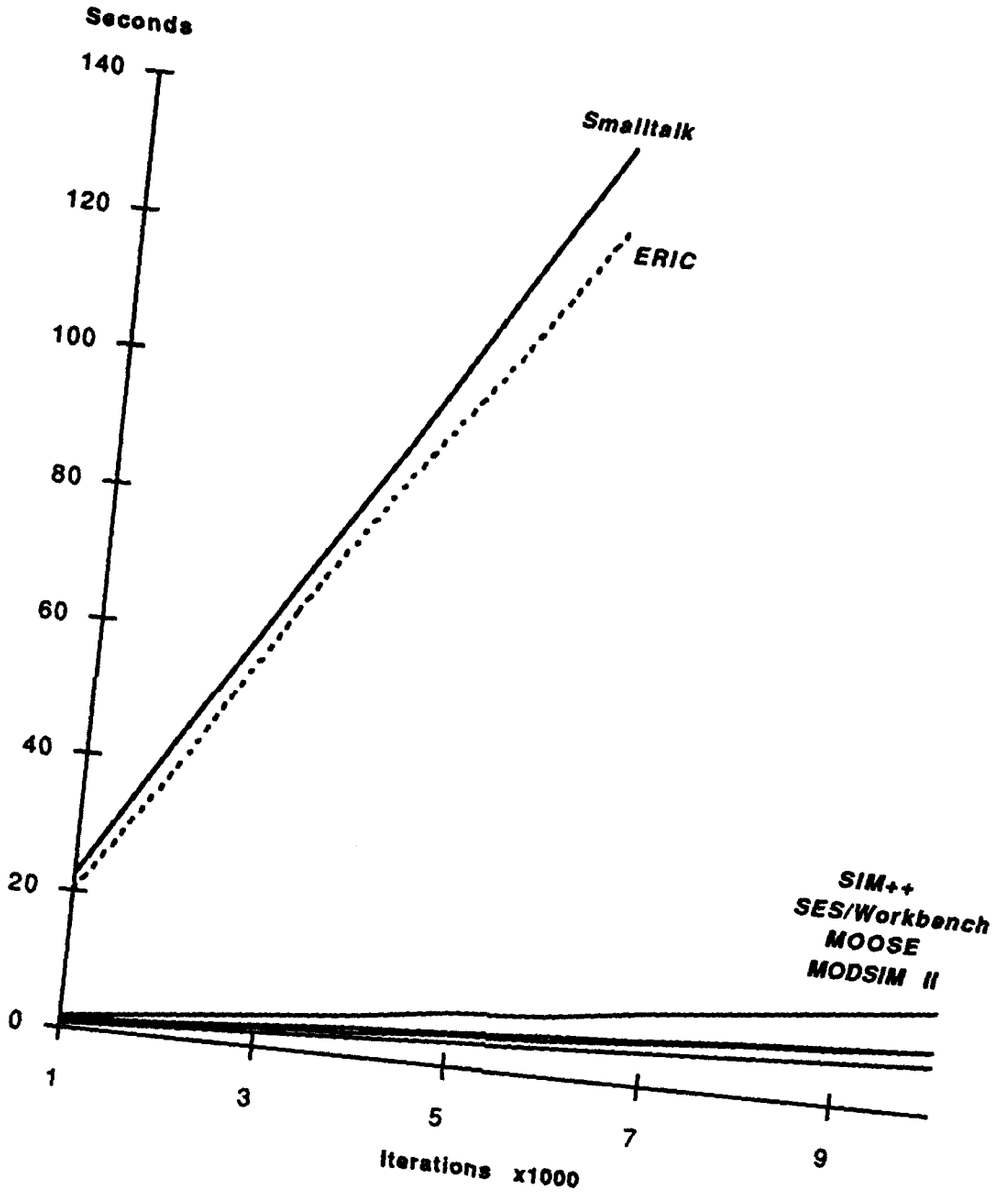


Figure 3. Test 2 Performance

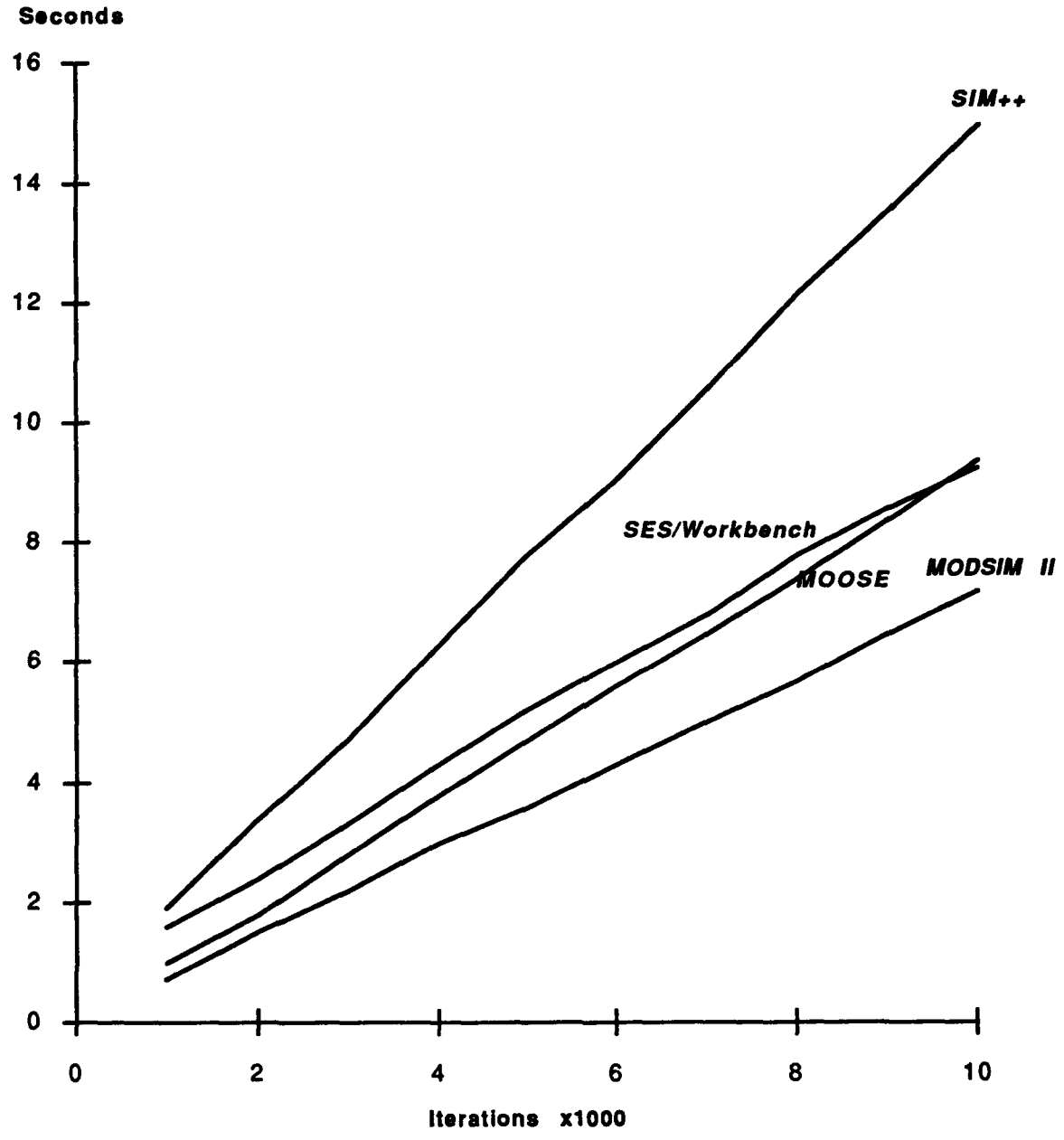


Figure 4. Test 2 Performance (no Smalltalk-80 or ERIC)

TEST 3

The results of benchmarks 3a and 3b are shown in figures 5 and 6, respectively.

MODSIM II is the only tool where a module must be both coded and called in a particular fashion, as a TELL method, if it might elapse simulation time. As the results of Test 3a show, there is a substantial performance penalty for calling a TELL method, using the WAIT FOR construct, even when the method does not elapse simulation time. The shape of the curve suggests that the quadratic sorting algorithm is invoked, as expected from Test 1.

Not surprisingly, the performance was best in the C++-based tools MOOSE and *Sim++*; the overhead introduced is that of a method invocation. Similarly, the Smalltalk implementation introduced a method invocation overhead which, while substantially more than that of the C++-based tools, was modest. The SES/*workbench* technique of invoking a subgraph was substantially slower than even the Smalltalk method invocation.

For all of the tools, the difference between Test 3a and Test 3b should have been measured by Test 2. This seems to be the case for all but Smalltalk, a result that is unexplained.

TEST 4

The results of this benchmark are shown in Figures 7 and 8. The results of the Smalltalk and *MODSIM II runs have been* omitted from figure 8 in order to better show the data for the others.

The overhead of acquiring a resource should be low. The implementation of a resource in the *Sim++* code shows the relatively simple operations needed and that a user can easily implement resources with a cost that is a small multiple of the cost of creating a thread. The nonlinear results of MODSIM II and Smalltalk are unexplained.

TEST 5

Figure 9 shows the results of this benchmark. Interrupts should incur an overhead equal to a small multiple of the cost of creating a thread. This expectation seems to be met by each of the tools.

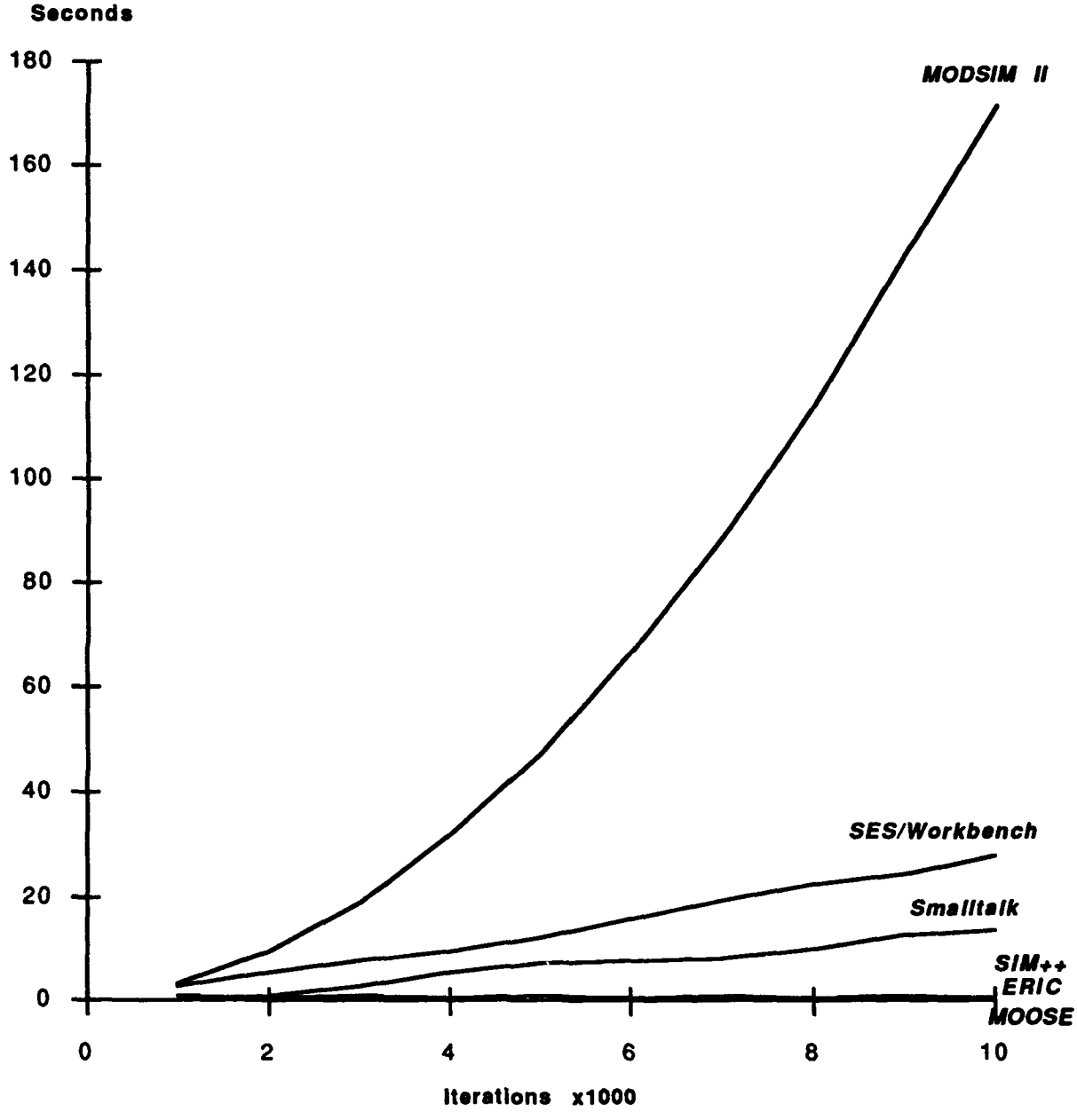


Figure 5. Test 3a Performance

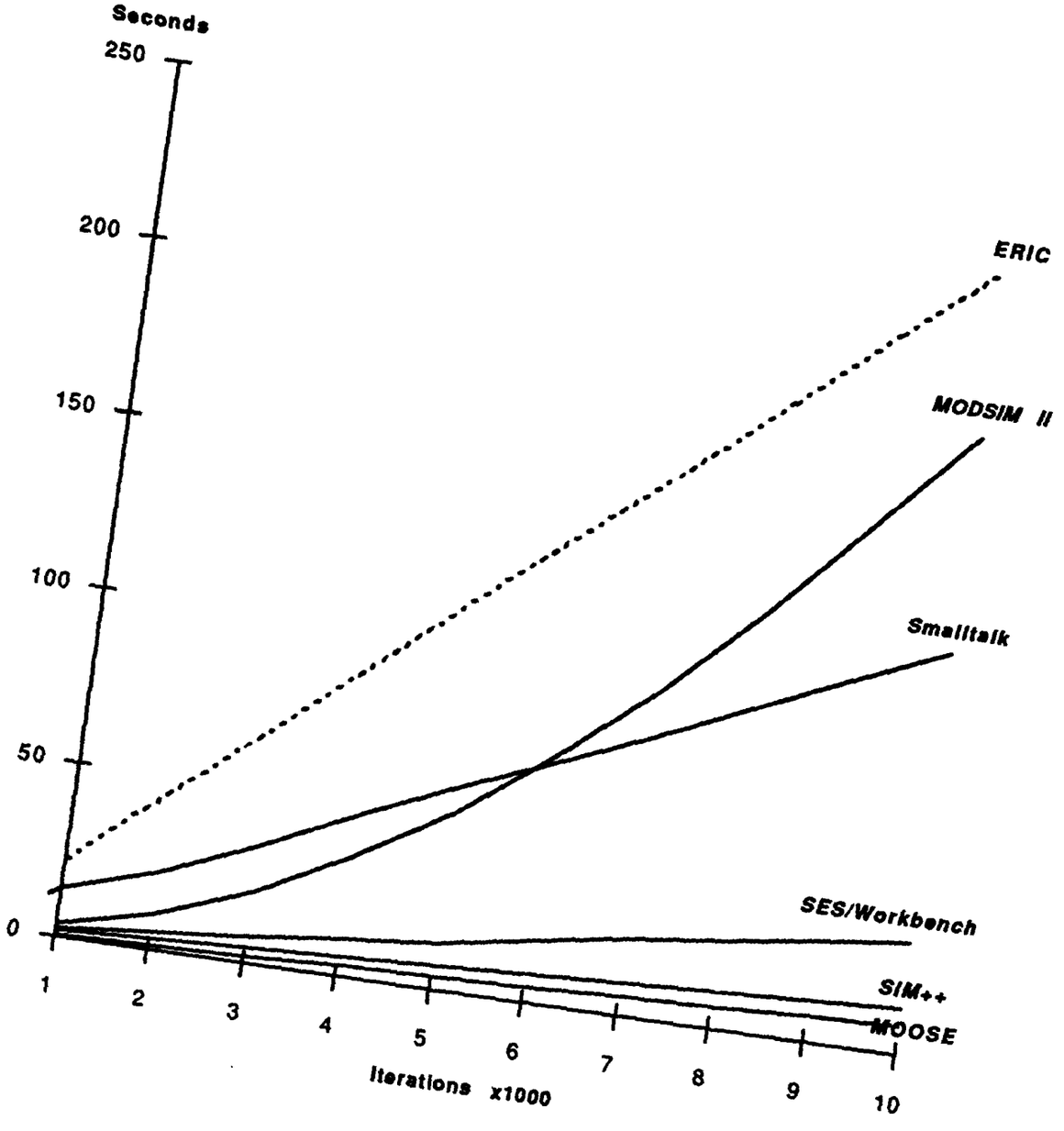


Figure 6. Test 3b Performance

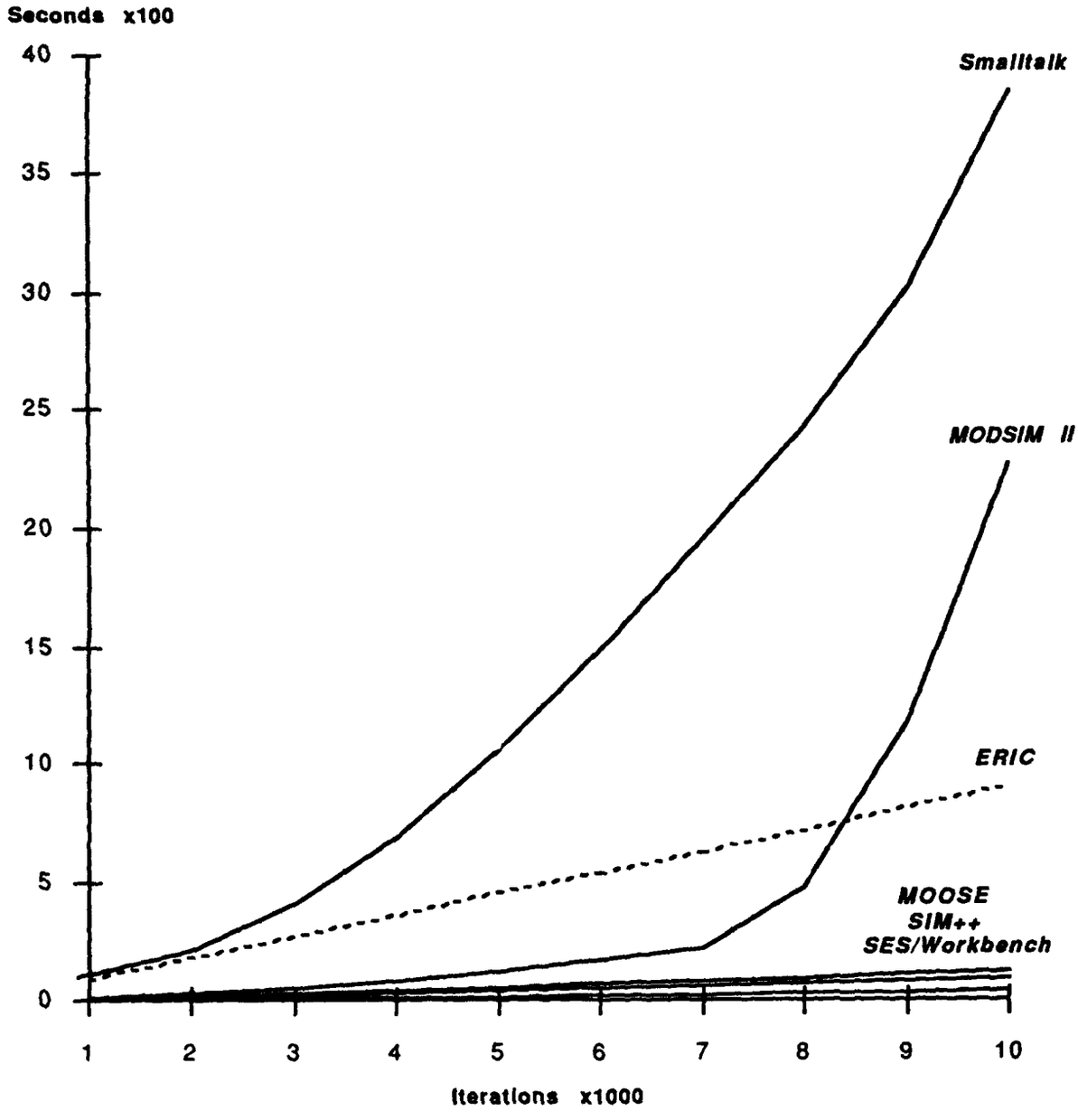


Figure 7. Test 4 Performance

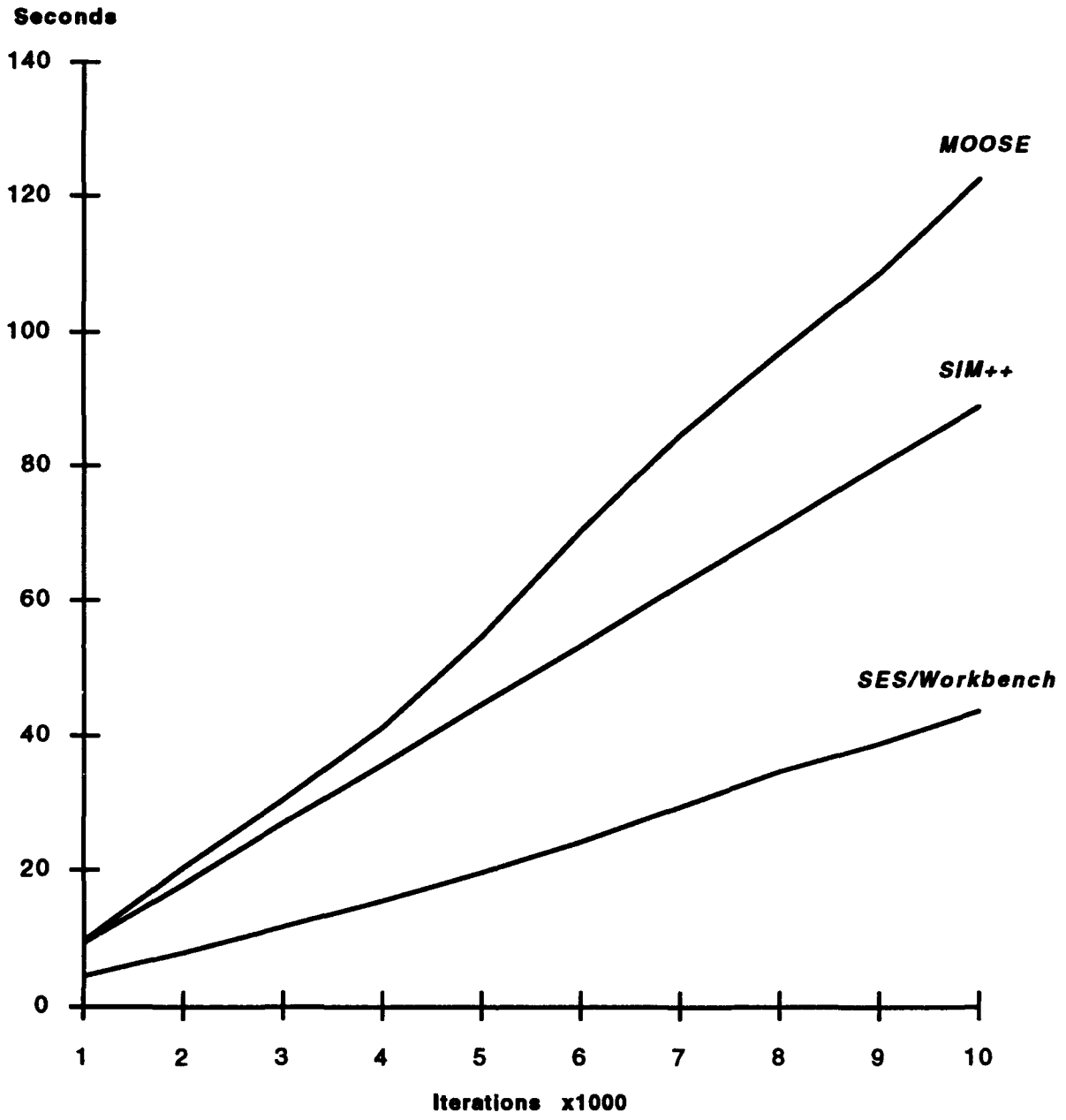


Figure 8. Test 4 Performance (Smalltalk-80, MODSIM II, and ERIC are omitted)

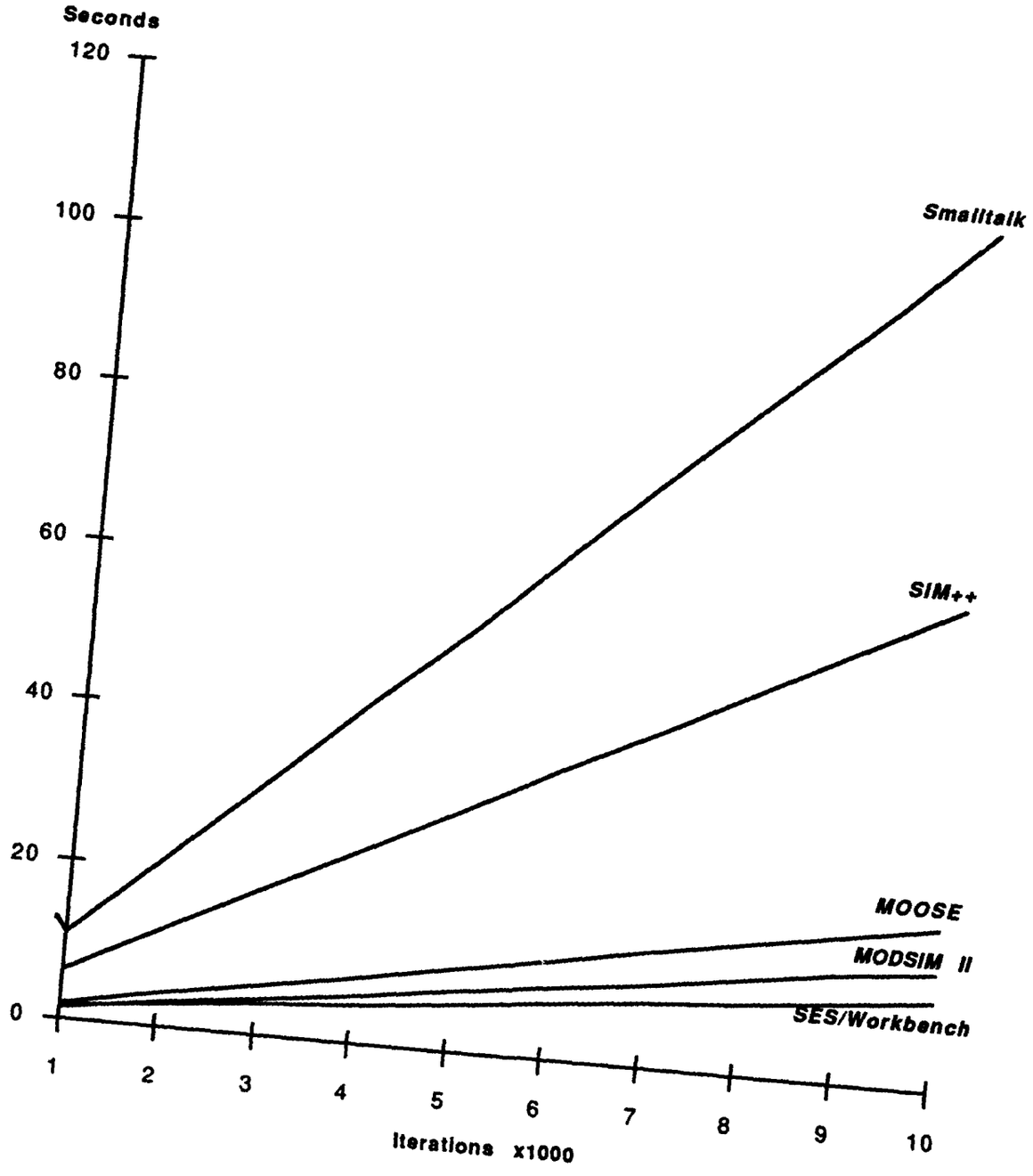


Figure 9. Test 5 Performance

BANK SIMULATION BENCHMARK

The results of the bank simulation benchmark are shown in figure 10. Two versions of this test were performed: one where the customer arrival rate closely matches the service times, yielding a system operating at its saturation point; in the second, the customer arrival rate exceeds the service capacity considerably, yielding a system operation well above its saturation point. The operation of the bank simulation is such that the queues never grow too long (customers leave to go to the restrooms, and leave the simulation entirely if the restroom lines are too long), so the difference in performance between the two versions of the test for each tool is not likely to be due to the queueing algorithms. Instead, the difference reflects the fact that each tool spends more of its time creating customer objects (threads) in the over saturated version than in the saturated version. For all tools except *Sim++*, thread creation is more expensive than other processing.

The overall results of the bank simulation test show that the relative performance of the tools on realistic simulation problems is predicted rather well by their relative performance on the single feature benchmarks.

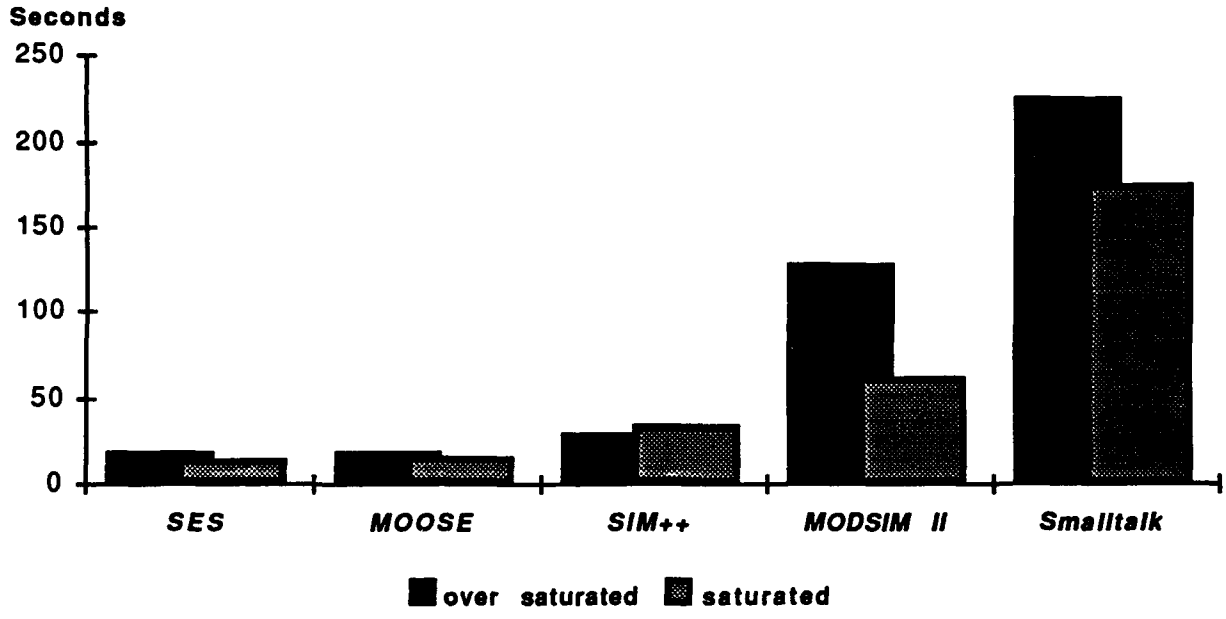


Figure 10. Bank simulation Performance

SECTION 6

SUMMARY AND CONCLUDING REMARKS

Our survey found a wide variety of features and performance in object-oriented simulation tools. On the performance side, Smalltalk represents one extreme, paying major penalties for making decisions at run time. At the other extreme is our hand-coded MOOSE, which excels at most of the benchmarks. Our benchmarking effort has generally reinforced the rule of thumb that Smalltalk code runs about one order of magnitude slower than comparable optimized C code. For the other systems that compile into C, we found SES/*workbench* to be surprisingly efficient, while MODSIM II and, to a lesser extent, *Sim++* were not as efficient.

It is important to emphasize that performance is only one facet of the evaluation of a simulation tool. In some contexts, the time and effort required to create the simulation code may be more critical than the code's execution time. While we are confident that our benchmarking effort provides clear performance distinctions, we feel less confident drawing conclusions concerning the time it takes to develop a simulation in a particular tool, or the time required to maintain or upgrade an existing simulation. These issues tend to depend on complex human factors. Some individuals my find the graphical interface of SES/*workbench* to be a large advantage, while others may feel that it is a hindrance in that it restricts access to the simulation code. The Smalltalk-80 language and programming environment provide a powerful collection of tools, but novice users will certainly be very bewildered during initial attempts to assimilate the system.

To consider the issue of programming languages in general, it is clear that C++ currently has a number of advantages: it is enjoying widespread popularity, with high quality and either public domain or low cost implementations available for an assortment of hardware platforms. Libraries of reusable classes for C++ are growing in number, and support for simulation is available from several sources. Environments that support C++ program development are becoming more widely used. The language has significant momentum and this is an important consideration when choosing a programming language.

The three commercial simulation tools are similar in their high licensing costs and their promise to provide users with support. In most other areas, these commercial tools are quite different. Let us start with MODSIM II. This tool has made a substantial amount of progress since its introduction. New features, such as compilation of circular references and WAIT FOR methods, have been introduced as users have identified problems. In addition, CACI has received preliminary benchmark information from us, and they have worked on the latest release of their system to improve their performance numbers. The basic simulation style of MODSIM II seems to be successful; we chose to exchange definitions of the bank simulation problem in MODSIM II code. The graphics support provided by CACI is certainly a positive feature. On the negative side, we have seen that the performance of MODSIM II is

disappointing. In terms of features, there are still a few things missing, for instance, overloading of method names. The decision to separate methods into ASK and TELL variants, and the subsequent addition of WAIT FOR methods, may not be the best path; a single type of method was adopted for MOOSE.

SES/*workbench* was included in this study since it is a tool currently available at MITRE and it makes some claims towards being object-oriented. In truth, the typical user interaction with this tool will involve none of the features of object-oriented programming. The support for object-oriented programming in the programming language SES/sim does not extend into the graphical interface. The clear focus in this product is extended queueing networks, and this is an important and very useful paradigm. SES has informed us that they are interested in working on a new tool that would be more object-oriented, but no details have been established. The performance of SES/*workbench* was impressive, especially given that the code was generated from graphic input.

Sim++ is a system specifically focused on parallel execution of simulations as a means of greatly improving performance. Our tests used only sequential *Sim++*, partly because the parallel version was not available to us, and partly because performance evaluation of parallel processing is outside the scope of what we could accomplish in this evaluation task. From the coding of the bank simulation, it seems fair to say that *Sim++* was the most difficult of our five primary systems to develop code in. This is due to that fact that the design methodology enforces an approach that facilitates parallel execution, but puts somewhat of an extra burden on the programmer. The performance of *Sim++* on the bank simulation was fairly good; it appears that the performance penalty created by the focus on parallel execution is less than the design penalty. Clearly, the utility of *Sim++* must be based on how successful it is at generating speed up when running in parallel. This would be an interesting topic for another performance study.

To conclude, as we began this project, we found little existing work in performance analysis for object-oriented systems. Our efforts have provided a start in this area. For simulation tools, we developed a small set of feature benchmarks. These benchmarks are certainly not exhaustive, and more work is necessary to assemble a complete approach to such benchmarks. Our single larger benchmark was of a rather simple system. It would be interesting to look at a more complex simulation. We believe it likely the performance characteristics of the tools would remain the same in a larger benchmark. One of the advantages of doing a larger benchmark would be to get more information on program development time.

LIST OF REFERENCES

Agre, J. R., and P. A.Tinker, January 1991, "Useful Extensions to a Time Warp Simulation System," *Proceedings of the 1991 SCS Conference on Parallel and Distributed Simulation*, Anaheim, CA, pages 78–85.

Bézivin, J., October 1987, "Some Experiments in Object-Oriented Simulation," *Proceedings of OOPSLA 87*, Kissimee, FL, pages 394–405.

Belanger, R., December 1990, "MODSIM II – A Modular, Object-Oriented Language," *Proceedings of the 1990 Winter Simulation Conference*, New Orleans, pages 118–122.

Belanger, R., B. Donovan, K. Morse, and D. Rockower, 1990, *MODSIM II, The Language for Object-oriented Programming: Reference Manual*, La Jolla, CA: CACI Products Company.

Bensley, E. H., V. T. Giddings, J. I. Leivent, and R. J. Watro, January 1992, "A Performance-based Comparison of Object-oriented Simulation Tools," *Proceedings of Object Oriented Simulation 1992*, Newport Beach, CA, pages 47–51.

Borning, A. H., and Ingalls, D. H. H., 1982, Multiple inheritance in Smalltalk-80, *Proceedings of AAAI, 1982*, Pittsburg, PA, pp. 234–237.

Chambers, C., D. Ungar, and E. Lee, October 1989, "An Efficient Implementation of Self, a Dynamically-typed Object-oriented Language Based on Prototypes, *Proceedings of OOPSLA 89*, New Orleans, LA, pages 49–70.

Delcambre, L. M. L., S. P. Landry, L. Pollacia, and J. Waramahaputi, January 1990, "Specifying Object Flow in an Object-Oriented Database for Simulation", *Object Oriented Simulation: Proceedings of the SCS Multiconference on Object Oriented Simulation*, San Diego, CA., pages 75-80.

Doyle, R. J., January 1990, "Object-oriented Simulation Programming", *1990 SCS Conference on Object Oriented Simulation*, pages 1– 6.

Eldredge, D. L., J. D. McGregor, and M. K. Summers, February 1990, "Applying the Object-oriented Paradigm to Discrete Event Simulations Using the C++ Language", *Simulation*, pages 83–91.

Fishman, G. S., 1973, *Concepts and Methods in Discrete Event Simulation*, New York: John Wiley and Sons.

Goldberg, A., and D. Robson, 1983, *Smalltalk-80: The Language and its Implementation*, Reading, MA: Addison-Wesley.

Herring, C., January 1990, "ModSim: A New Object-oriented Simulation Language", *1990 SCS Conference on Object Oriented Simulation*, pages 55–60.

Hilton, M. L., and J. D. Grimshaw, April 1990, ERIC Manual, RADC-TR-90-84.

Hooper, J. W., April 1986, "Strategy-Related Characteristics of Discrete-Event Languages and Models", *Simulation*, vol. 46, no. 4, pages 153–159.

Iacobovici, S., and C. Ng, August 1987, "VLSI and System Performance Modeling", *IEEE Micro*, pages 59–72.

Jade Simulations International, 1990, *Sim++ Programmer Reference Manual, Release 3.0,* Calgary, Canada: Jade Inc.

Kiviat, P. J., 1971, "Simulation Languages," *Computer Simulation Experiments With Models of Economic Systems,* (T. H. Naylor, ed.) New York, NY: John Wiley and Sons, pages 406–489.

Lippman, S. B., 1991, *C++ Primer*, 2nd editon, Reading, MA: Addison-Wesley.

Lomow, G., and D. Baezner, December 1990, "A Tutorial Introduction to Object-Oriented Simulation and *Sim++*," *Proceedings of the 1990 Winter Simulation Conference*, New Orleans, pages 149–153.

Schwetman, H. D., December 1990, "Introduction to Process-Oriented Simulation and CSIM," *Proceedings of the 1990 Winter Simulation Conference*, New Orleans, pages 154-157.

Scientific and Engineering Software, Inc, April 1989, *SES/workbench: Introductory Overview, Release 1.0*, Austin, TX: SES, Inc.

Stroustrup, B., 1991, *The C++ Programming Language*, 2nd Edition, Reading, MA: Addison-Wesley.

APPENDIX A

MODSIM II CODE

```
MAIN MODULE Test1;

FROM UtilMod IMPORT GetCmdLineArg;
FROM SimMod IMPORT StartSimulation, SimTime;
FROM RandMod IMPORT RandomObj;

TYPE
     Foo = OBJECT
           TELL METHOD Bar ();
     END OBJECT;

OBJECT Foo;
     TELL METHOD Bar ();
     BEGIN
     END METHOD;
END OBJECT;

VAR
     f : Foo;
     i, j : INTEGER;
     s : STRING;
     r : RandomObj;

BEGIN
     NEW(f);
     NEW(r);
     GetCmdLineArg(1, s);
     i := STRTOINT(s);
     FOR j := 1 TO i
           TELL f TO Bar() IN ASK r UniformReal(0.0,
1000.0);
     END FOR;
     StartSimulation();
END MODULE.
```

```
MAIN MODULE Test2;

FROM UtilMod IMPORT GetCmdLineArg;
FROM SimMod IMPORT StartSimulation, SimTime;
FROM RandMod IMPORT RandomObj;

TYPE
     Foo = OBJECT
           TELL METHOD Bar (IN n : INTEGER);
     END OBJECT;

VAR
     f : Foo;
     i, j : INTEGER;
     s : STRING;
     r : RandomObj;

OBJECT Foo;
     TELL METHOD Bar (IN n : INTEGER);
     BEGIN
           n := n - 1;
           IF n > 0
           TELL f TO Bar(n);
      END IF;
     END METHOD;
END OBJECT;

BEGIN
     NEW(f);
     NEW(r);
     GetCmdLineArg(1, s);
     i := STRTOINT(s);
     TELL f TO Bar(i);
     StartSimulation();
END MODULE.
```

```
MAIN MODULE Test3a;

FROM UtilMod IMPORT GetCmdLineArg;
FROM SimMod IMPORT StartSimulation, SimTime;
FROM RandMod IMPORT RandomObj;

TYPE
     Foo = OBJECT
           TELL METHOD Bar (IN n : INTEGER);
     END OBJECT;

VAR
     f : Foo;
     i, j : INTEGER;
     s : STRING;
     r : RandomObj;

OBJECT Foo;
     TELL METHOD Bar (IN n : INTEGER);
     BEGIN
           n := n - 1;
           IF n > 0
           WAIT FOR f TO Bar(n)
           END WAIT;
      END IF;
     END METHOD;
END OBJECT;

BEGIN
     NEW(f);
     NEW(r);
     GetCmdLineArg(1, s);
     i := STRTOINT(s);
     TELL f TO Bar(i);
     StartSimulation();
END MODULE.
```

```
MAIN MODULE Test3b;

FROM UtilMod IMPORT GetCmdLineArg;
FROM SimMod IMPORT StartSimulation, SimTime;
FROM RandMod IMPORT RandomObj;

TYPE
     Foo = OBJECT
           TELL METHOD Bar (IN n : INTEGER);
     END OBJECT;

VAR
     f : Foo;
     i, j : INTEGER;
     s : STRING;
     r : RandomObj;

OBJECT Foo;
     TELL METHOD Bar (IN n : INTEGER);
     BEGIN
      WAIT DURATION 1.0 END WAIT;
           n := n - 1;
           IF n > 0
           WAIT FOR f TO Bar(n)
           END WAIT;
      END IF;
     END METHOD;
END OBJECT;

BEGIN
     NEW(f);
     NEW(r);
     GetCmdLineArg(1, s);
```

```
    i := STRTOINT(s);
    TELL f TO Bar(i);
    StartSimulation();
END MODULE.



MAIN MODULE Test4;

FROM UtilMod IMPORT GetCmdLineArg;
FROM ResMod  IMPORT ResourceObj;
FROM SimMod  IMPORT StartSimulation, SimTime;

TYPE

    CustomerObj = OBJECT
          TELL METHOD Run(IN n : INTEGER);
    END OBJECT;

VAR

    Cust : CustomerObj;
    Res  : ResourceObj;
    I : INTEGER;
    Str  : STRING;

OBJECT CustomerObj;
    TELL METHOD Run(IN n : INTEGER);
    BEGIN
          IF n > 1
                TELL SELF TO Run(n - 1) IN 1.0;
      END IF;
          WAIT FOR Res TO Give(SELF,1);
          END WAIT;
          WAIT DURATION 1000000.0
          END WAIT;
          ASK Res TO TakeBack(SELF,1);
    END METHOD;
END OBJECT;

BEGIN
    NEW(Cust);
    NEW(Res);
    ASK Res TO Create(1);
    GetCmdLineArg(1, Str);
    I := STRTOINT(Str);
    TELL Cust TO Run(I);
    StartSimulation;
END MODULE.



MAIN MODULE Test5;

FROM UtilMod IMPORT GetCmdLineArg;
FROM SimMod IMPORT StartSimulation, SimTime,
Interrupt;

TYPE
    Foo = OBJECT
          TELL METHOD LongDelayLoop (IN I: INTEGER);
         TELL METHOD InterruptLoop (IN J: INTEGER);
    END OBJECT;

OBJECT Foo;

    TELL METHOD LongDelayLoop (IN I: INTEGER);
    BEGIN
    (OUTPUT("LongDelayLoop started");)
    WAIT DURATION 100.0
    ON INTERRUPT
       (OUTPUT("LongDelay interrupted");)
    IF I > 0 TELL SELF TO LongDelayLoop(I - 1);
       END IF;
       END WAIT;
    (OUTPUT("LongDelayLoop finished");)
    END METHOD;

    TELL METHOD InterruptLoop (IN J: INTEGER);
    BEGIN
    (OUTPUT("Interrupter started");)
    WAIT DURATION 1.0
    END WAIT;
    IF J > 0 Interrupt(SELF, "LongDelayLoop");
       TELL SELF TO InterruptLoop(J - 1);
    END IF;
    (OUTPUT("Interrupter finished");)
    END METHOD;

END OBJECT;

VAR
    f    : Foo;
    Num : INTEGER;
    Str : STRING;

BEGIN
    NEW(f);
    GetCmdLineArg(1, Str);
    Num := STRTOINT(Str);
    TELL f TO LongDelayLoop(Num);
    TELL f TO InterruptLoop(Num);
    StartSimulation();
END MODULE.



MAIN MODULE Sim;

FROM SimMod IMPORT StartSimulation, SimTime,
Interrupt, TriggerObj;
FROM ResMod IMPORT ResourceObj;
FROM RandMod IMPORT RandomObj;
FROM MathMod IMPORT LN;
FROM UtilMod IMPORT ClockTimeSecs;

TYPE
    Gender = (Female, Male);

    CustomerObj = OBJECT; FORWARD;

    VIPObj = OBJECT; FORWARD;

    LineObj = OBJECT(ResourceObj)
          serving : CustomerObj;
      canContinue : TriggerObj;
          TELL METHOD ServeCust (IN cust :
CustomerObj);
          OVERRIDE
               ASK METHOD ObjInit ();
    END OBJECT;

    RestRoom = ResourceObj;

    CustomerObj = OBJECT
          myGender : Gender;
          ASK METHOD ObjInit ();
          TELL METHOD GetOnLine ();
          TELL METHOD VisitFacilities ();
          PRIVATE
           ASK METHOD FindBestLine () : LineObj;
    END OBJECT;

    EndSim = OBJECT
          TELL METHOD Stop ();
    END OBJECT;

    VIPObj = OBJECT(CustomerObj)
          OVERRIDE
           TELL METHOD GetOnLine ();
    END OBJECT;

    CustGeneratorObj = OBJECT
          TELL METHOD GenCustomers ();
    END OBJECT;
```

```
     VIPGeneratorObj = OBJECT(CustGeneratorObj)
          OVERRIDE
               TELL METHOD GenCustomers ();
     END OBJECT;

     MoreRandomObj = OBJECT(RandomObj)
          { add the Erlang distrubution }
          ASK METHOD Erlang (IN mean, variance :
REAL) : REAL;
     END OBJECT;


VAR
     hoursToRun,
     meanInterArriveTime,
     meanVIPInterArriveTime,
     meanNatureCallsTime,
     varianceNatureCallsTime,
     meanLineTolerance,
     varianceLineTolerance,
     meanServiceTime,
     varianceServiceTime : REAL;
     k : INTEGER;
     numLines : INTEGER;
     line : LineObj;
     allLines : ARRAY INTEGER OF LineObj;
     random : MoreRandomObj;
     restRooms : ARRAY Gender OF RestRoom;
     restRoomMeanTime,
     restRoomVarTime : ARRAY Gender OF REAL;
     custGenerator : CustGeneratorObj;
     vipGenerator : VIPGeneratorObj;
     seed : INTEGER;
     endsim : EndSim;

OBJECT CustomerObj;
     ASK METHOD ObjInit ();
     BEGIN
          myGender := VAL(Gender, ASK random
UniformInt(0, 1));
       TELL SELF TO GetOnLine;
     END METHOD;

     TELL METHOD GetOnLine ();
     VAR
          myLine : LineObj;
       timeTillNatureCalls : REAL;
     BEGIN
          LOOP
               LOOP
                    timeTillNatureCalls := ASK
random

Erlang(meanNatureCallsTime,

varianceNatureCallsTime);
                    myLine := ASK SELF TO
FindBestLine();
                    WAIT FOR myLine TO
TimedGive(SELF, 1, timeTillNatureCalls)
                         EXIT;
                    ON INTERRUPT
                         WAIT FOR SELF TO
VisitFacilities
                  END WAIT;
                    END WAIT;
          END LOOP;
          WAIT FOR myLine TO ServeCust(SELF)
          END WAIT;
          ASK myLine TO TakeBack(SELF, 1);
          EXIT;
       END LOOP;
       DISPOSE(SELF);
     END METHOD;

     TELL METHOD VisitFacilities ();
     VAR
          restRoom : RestRoom;
       restRoomLineTolerance : INTEGER;
     BEGIN
          restRoom := restRooms[myGender];
       restRoomLineTolerance := ROUND(ASK random
Erlang(meanLineTolerance,

varianceLineTolerance));
          IF ASK restRoom TO ReportNumberPending() >
restRoomLineTolerance
            DISPOSE(SELF);
            TERMINATE;
       ELSE
            WAIT FOR restRoom TO Give(SELF, 1)
            END WAIT;
            WAIT DURATION ASK random
Erlang(restRoomMeanTime[myGender],

restRoomVarTime[myGender])
            END WAIT;
            ASK restRoom TO TakeBack(SELF, 1);
       END IF;
     END METHOD;

     ASK METHOD FindBestLine () : LineObj;
     VAR
          line, bestLine : LineObj;
       length, bestLength, i : INTEGER;
     BEGIN
          bestLength := MAX(INTEGER);
       FOR i := 1 TO numLines
            line := allLines[i];
            length := ASK line TO
ReportNumberPending();
            IF ASK line TO ReportAvailability() = 0
                 length := length + 1;
            END IF;
            IF length < bestLength
                 bestLength := length;
              bestLine := line;
            END IF;
       END FOR;
       RETURN bestLine;
     END METHOD;
END OBJECT;

OBJECT VIPObj;
     TELL METHOD GetOnLine ();
     VAR
          line : LineObj;
       oldCust : CustomerObj;
     BEGIN
          line := allLines[ASK random UniformInt(1,
numLines)];
       oldCust := ASK line serving;
       IF oldCust <> NILOBJ
            IF ISANCESTOR(VIPObj, oldCust)
                 RETURN;
            END IF;
              Interrupt(line, "ServeCust");
       ELSE
            WAIT FOR line TO Give(SELF, 1)
            END WAIT;
       END IF;
       WAIT FOR line TO ServeCust(SELF)
       END WAIT;
       IF oldCust <> NILOBJ
            WAIT FOR line.canContinue TO Trigger()
            END WAIT;
       ELSE
            ASK line TO TakeBack(SELF, 1);
       END IF;
       DISPOSE(SELF);
     END METHOD;
END OBJECT;

OBJECT MoreRandomObj;
     ASK METHOD Erlang (IN mean, variance : REAL) :
REAL;
     VAR
          k : INTEGER;
       i : INTEGER;
       prod : REAL;
     BEGIN
          k := ROUND(mean * mean / variance);
```

```
      IF k <= 0
           OUTPUT("Bad parameters to Erlang.");
           HALT;
      END IF;
      prod := 1.0;
      FOR i := 1 TO k
           prod := prod * ASK SELF UniformReal(0.0,
1.0);
      END FOR;
      RETURN -LN(prod) * mean / FLOAT(k);
    END METHOD;
END OBJECT;

OBJECT LineObj;
    ASK METHOD ObjInit ();
    BEGIN
         INHERITED ObjInit;
         NEW(canContinue);
    END METHOD;

    TELL METHOD ServeCust (IN cust : CustomerObj);
    VAR
         svcTime : REAL;
      startTime : REAL;
    BEGIN
         svcTime := ASK random
Erlang(meanServiceTime, varianceServiceTime);
      LOOP
           startTime := SimTime();
           serving := cust;
           IF svcTime <= 0.0
                EXIT;
           END IF;
           WAIT DURATION svcTime
             serving := NILOBJ;
             EXIT;
           ON INTERRUPT
                svcTime := svcTime - (SimTime()
- startTime);
            LOOP
                WAIT FOR canContinue TO Fire()
                     EXIT;
                ON INTERRUPT
                END WAIT;
            END LOOP;
           END WAIT;
      END LOOP;
    END METHOD;
END OBJECT;

OBJECT CustGeneratorObj;
    TELL METHOD GenCustomers();
    VAR
         customer  : CustomerObj;
         waitTime  : REAL;
    BEGIN
         LOOP
              waitTime := ASK random
Exponential(meanInterArriveTime);
              WAIT DURATION waitTime;
              END WAIT;
              NEW(customer);
         END LOOP;
    END METHOD;  { GenCustomers }
END OBJECT;

OBJECT VIPGeneratorObj;
    TELL METHOD GenCustomers();
    VAR
         vip : VIPObj;
         waitTime  : REAL;
    BEGIN
         IF meanVIPInterArriveTime > 0.0
              LOOP
                   waitTime := ASK random
Exponential(meanVIPInterArriveTime);
                   WAIT DURATION waitTime;
                   END WAIT;
                   NEW(vip);
              END LOOP;
      END IF;
    END METHOD;  { GenCustomers }
END OBJECT;

OBJECT EndSim;
    TELL METHOD Stop ();
    BEGIN
         HALT();
    END METHOD;
END OBJECT;

BEGIN
    {NEW(dt);}
    {TELL dt TO TimeOut() IN 1.0;}
    NEW(restRooms, Female .. Male);
    NEW(restRooms[Female]);
    ASK restRooms[Female] TO Create(4);
    NEW(restRooms[Male]);
    ASK restRooms[Male] TO Create(4);
    NEW(restRoomMeanTime, Female .. Male);
    NEW(restRoomVarTime, Female .. Male);
    restRoomMeanTime[Female] := 5.0;
    restRoomMeanTime[Male] := 3.0;
    restRoomVarTime[Female] := 8.0;
    restRoomVarTime[Male] := 6.0;
    OUTPUT("MODSIM II Simulation 'Lines and Rest
Rooms' starting--");
    OUTPUT("What is the mean customer interarrival
time in minutes?");
    INPUT(meanInterArriveTime);
    OUTPUT("What is the mean VIP interarrival time in
minutes?");
    INPUT(meanVIPInterArriveTime);
    OUTPUT("What is the mean service time in
minutes?");
    INPUT(meanServiceTime);
    OUTPUT("What is the variance of the service
time?");
    INPUT(varianceServiceTime);
    OUTPUT("What is the mean time in minutes till
'Nature Calls'?");
    INPUT(meanNatureCallsTime);
    OUTPUT("What is the variance?");
    INPUT(varianceNatureCallsTime);
    OUTPUT("What is the mean restroom line length
tolerance?");
    INPUT(meanLineTolerance);
    OUTPUT("What is the variance?");
    INPUT(varianceLineTolerance);
    OUTPUT("How many lines are there?");
    INPUT(numLines);
    OUTPUT("How many hours should the simulation
run?");
    INPUT(hoursToRun);
    OUTPUT("Random Seed?");
    INPUT(seed);

    OUTPUT("Mean Interarrive Time: ",
meanInterArriveTime);
    OUTPUT("Mean VIP Interarrive Time: ",
meanVIPInterArriveTime);
    OUTPUT("Mean Service Time: ", meanServiceTime);
    OUTPUT("Variance Service Time: ",
varianceServiceTime);
    OUTPUT("Mean 'Nature Calls' Time: ",
meanNatureCallsTime);
    OUTPUT("Variance 'Nature Calls' Time: ",
varianceNatureCallsTime);
    OUTPUT("Mean line length tolerance: ",
meanLineTolerance);
    OUTPUT("Variance line length tolerance: ",
varianceLineTolerance);
    OUTPUT("Number of lines: ", numLines);
    OUTPUT("Hours to run: ", hoursToRun);
    NEW(allLines, 1 .. numLines);
    FOR k := 1 TO numLines
         NEW(line);
      ASK line TO Create(1);
      allLines[k] := line;
    END FOR;
    NEW(random);
    ASK random TO SetSeed(seed);
    NEW(custGenerator);
    NEW(vipGenerator);
```

```
    NEW(endsim);
    TELL custGenerator TO GenCustomers();
    TELL vipGenerator TO GenCustomers();
    TELL endsim TO Stop IN 60.0 * hoursToRun;
    StartSimulation;
END MODULE.
```

APPENDIX B

SES/*workbench* CODE

On the following three pages, we have reproduces screen images of the directed graphs that represent the top level of the benchmarks in SES/*workbench*. Given the previous descriptions of the benchmarks, these graphs should provide enough detail to understand how the benchmarks were realized in the SES tool.

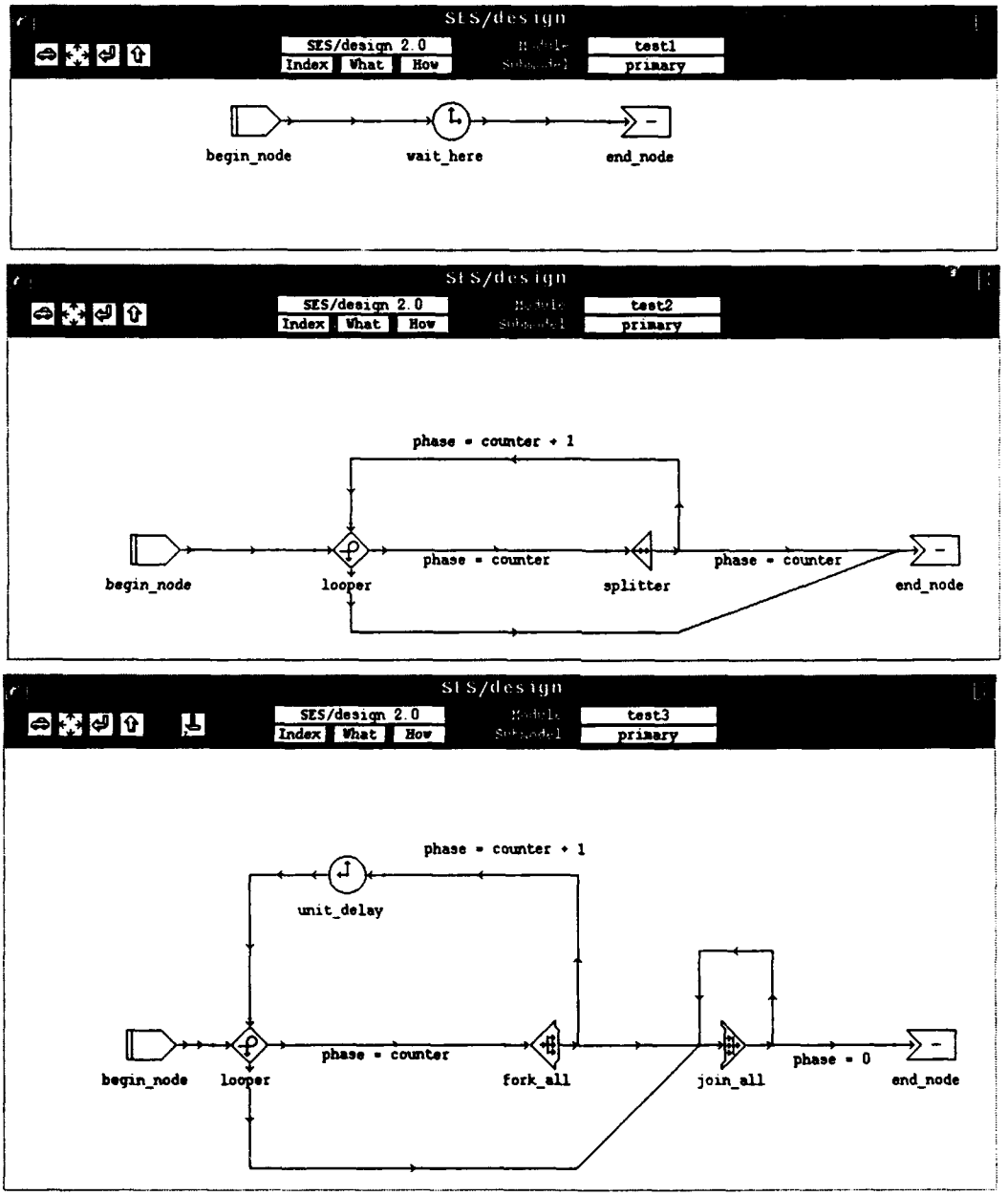


Figure 11. SES/*workbench* Graphs

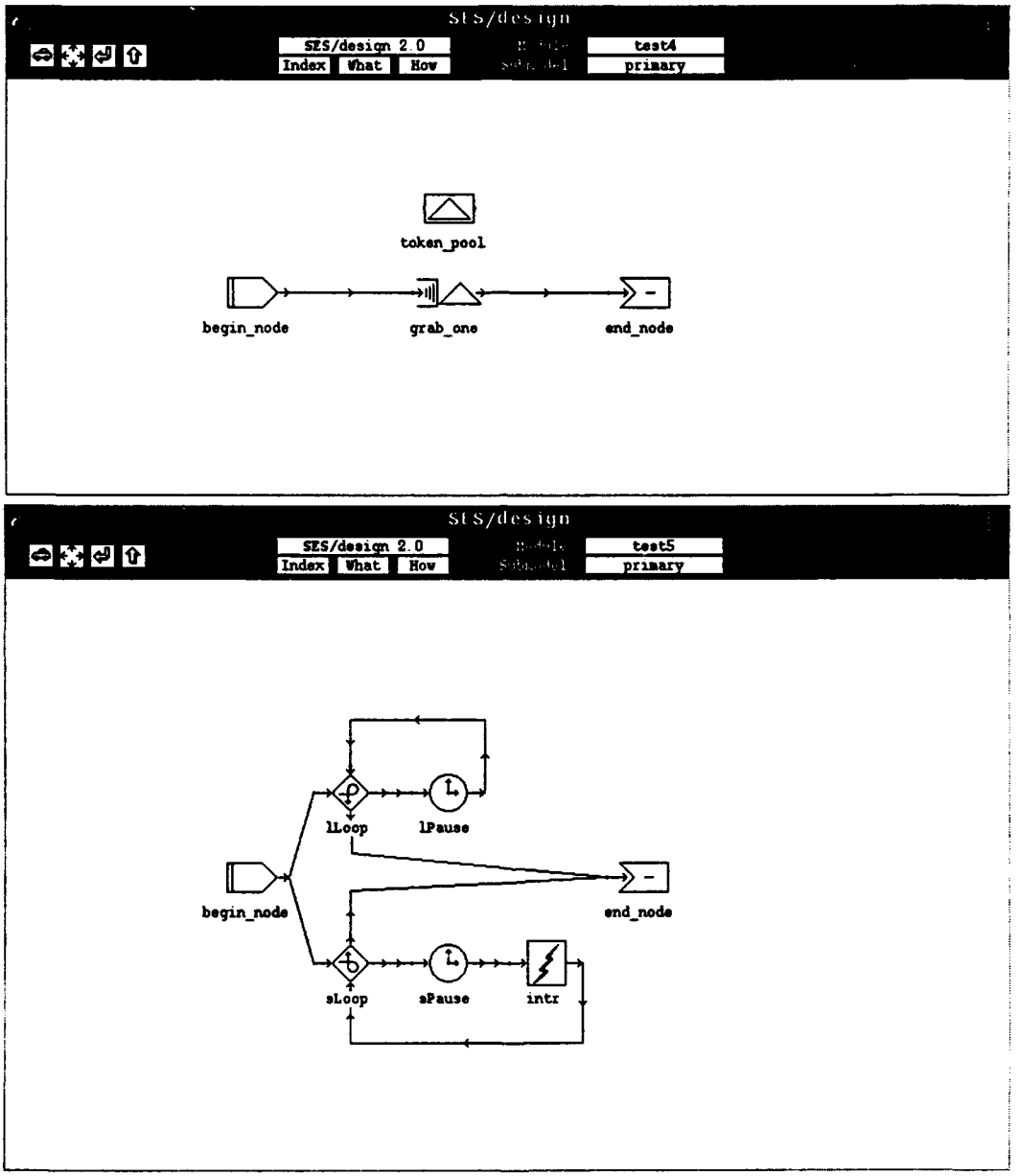
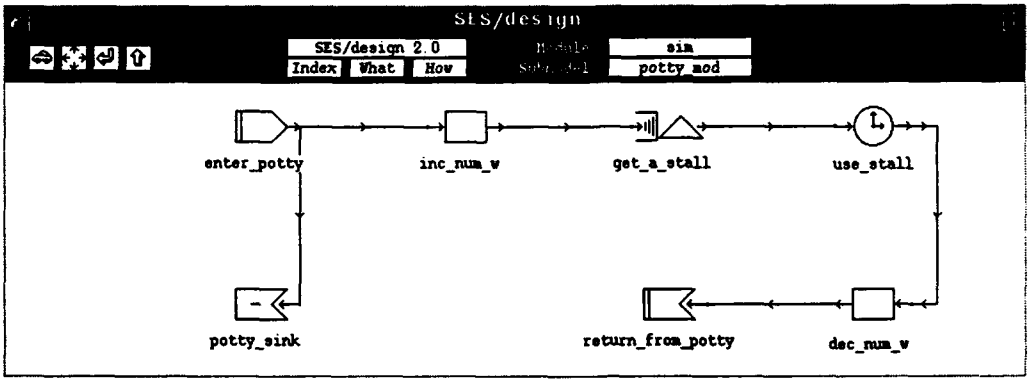


Figure 11. (continued)



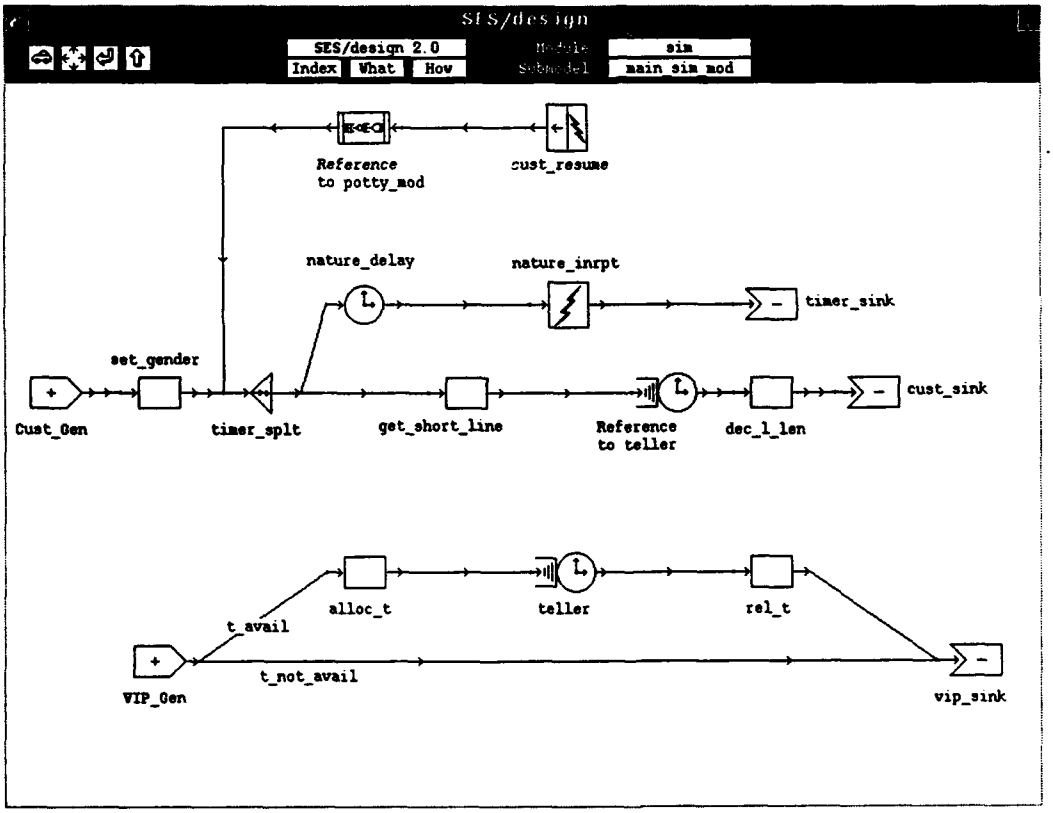


Figure 11. (concluded)

APPENDIX C

Sim++ CODE

```
//////// Test1.c  -- test1 (1 queue)
#include <sim++.h>
#include <stdlib.h>

// message types
enum ( INIT, GO, STOP);

// receiver entity class
class foo : public sim_entity {
   public:
     foo( sim_event &ev);
     void body();
     void bar();
};


foo::foo( sim_event &ev)
{
  // initialize the entity
  }

void foo::body()
{
   sim_event ev;
   sim_wait( ev);
   while ( ev.type() != STOP ) {
     bar();
     sim_wait( ev);
   }
 }

void foo::bar() {
}


// instantiate entity class
SIM_ENTITY(foo);

// id for entity instantiation
sim_entity_id foo_id;

// source entity class
class source : public sim_entity {
   public:
      source( sim_event &ev);
      void body();
   private:
      int n;
};

source::source( sim_event &ev)
{
   // get the number of iterations
   SIM_GET( int, n, ev);
}

void source::body()
{

     // Put n events on queue.
     int seed = 12345;
     double time;
     while (n--)
       {
          time = sim_uniform( 1.0, 1000.0, seed);
       sim_schedule(foo_id, time, GO);
     }

     // Schedule the stopping event
     sim_schedule( foo_id, 1001.0, STOP);
}

// instantiate entity class
SIM_ENTITY(source);

// id for entity instantiation
sim_entity_id source_id;


// create and initialize entities
void sim_initialize( int, char *argv[], int, char*[])
{
     int n = atoi(argv[1]);
     char *foo_name = "foo_entity";
     char *source_name = "source_entity";

     // create one instance of source class with ID
source_id and name
     // source_name
     source_id = sim_create("source",source_name,
INIT, SIM_PUT( int, n));

     // create one instance of foo class with ID
foo_id and name foo_name
     foo_id = sim_create( "foo", foo_name, INIT);

}



////////////// test1b.c -- test 1 (n queues)
////////////// note: must subtract the results
////////////// test1c in order to remove the
////////////// the overhead of creating
////////////// n entities
#include <sim++.h>
#include <stdlib.h>
#define MAX_ENTITIES 10000

// message types
enum ( INIT, GO, STOP);

// receiver entity class
class foo : public sim_entity {
   public:
     foo( sim_event &ev);
     void body();
     void bar();
};


foo::foo( sim_event &ev)
{
  // initialize the entity
  }

void foo::body()
{
   sim_event ev;
   sim_wait( ev);
   while ( ev.type() != STOP ) {
     bar();
     sim_wait( ev);
   }
 }

void foo::bar() {
}


// instantiate entity class
SIM_ENTITY(foo);

// id for entity instantiation
sim_entity_id foo_id[MAX_ENTITIES];
```

```
struct sizes {
   int m;
   int n;
};

// source entity class
class source : public sim_entity {
   public:
      source( sim_event &ev);
      void body();
   private:
      sizes size;
};

source::source( sim_event &ev)
{
   // get the number of iterations
   SIM_GET(sizes, size, ev);
}

int seed = 12345;

void source::body()
{

     // Put n events on queue
     double time;
     for (int n_events = size.n; n_events > 0;
n_events--) {
        for ( int m_entities = size.m; m_entities > 0;
m_entities--) {
           time = sim_uniform( 1.0, 1000.0, seed);
        sim_schedule(foo_id[m_entities], time, GO);
     }
     }

}

// instantiate entity class
SIM_ENTITY(source);

// id for entity instantiation
sim_entity_id source_id;


// create and initialize entities
void sim_initialize( int, char *argv[], int, char*[])
{
    sizes size;
    size.m = atoi(argv[1]);
    size.n = atoi(argv[2]);
     char foo_name[10];
     char *source_name = "source_entity";

    // create one instance of source class with name
source_entity
    sim_create("source",source_name, INIT,
SIM_PUT(sizes, size));

     // create "m" instances of foo class with name
foo_"i"
    for ( int i = 1; i <= size.m; i++) {
       sprintf(foo_name, "foo_%d", i);
       foo_id[i] = sim_create( "foo", foo_name, INIT);
   }

}
///////// test1c.c -- test1 (n queues)
///////// note: schedules no events,
///////// used to measure the overhead
///////// of creating n entities
#include <sim++.h>
#include <stdlib.h>
#define MAX_ENTITIES 10000

// message types
enum { INIT, GO, STOP};

// receiver entity class
class foo : public sim_entity {
   public:
     foo( sim_event &ev);
     void body();
     void bar();
};


foo::foo( sim_event &ev)
{
  // initialize the entity
  }

void foo::body()
{
   sim_event ev;
   sim_wait( ev);
   while ( ev.type() != STOP ) {
     bar();
     sim_wait( ev);
   }
 }

void foo::bar() {
}


// instantiate entity class
SIM_ENTITY(foo);

// id for entity instantiation
sim_entity_id foo_id[MAX_ENTITIES];

// source entity class
class source : public sim_entity {
   public:
      source( sim_event &ev);
      void body();
};

source::source( sim_event &ev)
{
}

void source::body()
{
}

// instantiate entity class
SIM_ENTITY(source);

// id for entity instantiation
sim_entity_id source_id;


// create and initialize entities
void sim_initialize( int, char *argv[], int, char*[])
{
    int m = atoi(argv[1]);
    int n = atoi(argv[2]);
     char foo_name[10];
     char *source_name = "source_entity";

    // create one instance of source class with name
source_entity
    sim_create("source",source_name, INIT);

     // create "m" instances of foo class with name
foo_"i"
    for ( int i = 1; i <= m; i++) {
       sprintf(foo_name, "foo_%d", i);
       foo_id[i] = sim_create( "foo", foo_name, INIT);
   }

}
////// test2.c
#include <sim++.h>
#include <stdlib.h>

// message types
enum { INIT, GO, STOP};

// id for entity instantiation
sim_entity_id foo_id;
```

```
// receiver entity class
class foo : public sim_entity {
   public:
     foo( sim_event &ev);
     void body();
   private:
     int n;
};


foo::foo( sim_event &ev)
{
   // initialize the entity
   SIM_GET( int, n, ev);
  }

void foo::body()
{
   sim_event ev;
   while ( --n )
      {
      // schedule an event for yourself
      sim_schedule(foo_id, 0.0, GO);
          // and wait for it
      sim_wait( ev);
     }
 }

// instantiate entity class
SIM_ENTITY(foo);

// create and initialize entities
void sim_initialize( int argc, char *argv[], int,
char*[])
{
     int n = atoi(argv[1]);
     char *foo_name = "foo_entity";

     // create one instance of class foo with ID
foo_id and name foo_name
     foo_id = sim_create( "foo", foo_name, INIT,
SIM_PUT( int, n));


}
////// test3a.c
#include <sim++.h>
#include <stdlib.h>

// message types
enum { INIT, GO, STOP};

// id for entity instantiation
sim_entity_id foo_id;

// receiver entity class
class foo : public sim_entity {
   public:
     foo( sim_event &ev);
     void body();
     void bar(int);
   private:
     int n;
};

foo::foo( sim_event &ev)
{
   SIM_GET( int, n, ev);
 }

void foo::body()
{
   bar (n);
 }

void foo::bar(int n) {
     /*sim_printf("bar: %d \n", n);*/
     if (--n == 0) {
          sim_event ev;
          sim_time new_time = sim_hold( 1.0, ev);
     }
     else
          bar(n);
}

// instantiate entity class
SIM_ENTITY(foo);

// create and initialize entities
void sim_initialize( int, char *argv[], int, char*[])
{
     int n = atoi(argv[1]);
     char *foo_name = "foo_entity";

     // create one instance of class foo id ID foo_id
and name foo_name
     foo_id = sim_create( "foo", foo_name, INIT,
SIM_PUT( int, n));

   }



///// test3b.c
#include <sim++.h>
#include <stdlib.h>

// message types
enum { INIT, GO, STOP};

// id for entity instantiation
sim_entity_id foo_id;

// receiver entity class
class foo : public sim_entity {
   public:
     foo( sim_event &ev);
     void body();
     void bar(int);
   private:
     int n;
};

foo::foo( sim_event &ev)
{
   SIM_GET( int, n, ev);
 }

void foo::body()
{
   bar (n);
 }

void foo::bar(int n) {
     sim_event ev;
     if (--n > 0) {
          sim_time new_time = sim_hold( 1.0, ev);
      bar(n);
      }
}

// instantiate entity class
SIM_ENTITY(foo);

// create and initialize entities
void sim_initialize( int, char *argv[], int, char*[])
{
     int n = atoi(argv[1]);
     char *foo_name = "foo_entity";

     // create one instance of class foo id ID foo_id
and name foo_name
     foo_id = sim_create( "foo", foo_name, INIT,
SIM_PUT( int, n));

   }

////////////////////////////// resources.h
#include <sim++.h>
```

```
// resource entity superclass
class resources : public sim_entity {
   public:
     // message types
     enum { INIT, GIVE, GIVE_ACK, TAKE_BACK, DESTROY};
     // initialize an instance
     resources( sim_event &ev);
     // simple behaviour
     void body();
     sim_event next_give_or_destroy();
     sim_event next_take_back_or_destroy(sim_entity_id
from);
};

// customer entity class
class consumers {
   public:
      void acquire_resource( sim_entity_id
&the_resource);
      void give_back_resource( sim_entity_id
&the_resource);
      void destroy_resource( sim_entity_id
&the_resource);
};

////////////////////// consumers.c
#include "resources.h";

void consumers::acquire_resource( sim_entity_id
&the_resource)
{
   const sim_type_p got_it( resources::GIVE_ACK);
   sim_event ev;
   sim_event_id event_id;

   event_id = sim_schedule( the_resource, 0.0,
resources::GIVE);
   sim_wait_for( got_it, ev);
 }

void consumers::give_back_resource( sim_entity_id
&the_resource)
{
   sim_event ev;
   sim_event_id event_id = sim_schedule( the_resource,
0.0,
      resources::TAKE_BACK);
}

void consumers::destroy_resource( sim_entity_id
&the_resource)
{
   sim_event_id event_id = sim_schedule( the_resource,
0.0,
      resources::DESTROY);
 }
////////////////// resources.c
#include "resources.h";

resources::resources( sim_event &ev)
{  // no need to initialize further
   }

sim_event resources::next_give_or_destroy()
{
   const sim_type_p give_or_destroy_p( GIVE, DESTROY);
   sim_event event_received;
   // try to select a deferred event that matches
   int bool = sim_select( SIM_ANY, event_received);
   if (bool == 0) {
      // no deferred events, wait for next GIVE or
DESTROY
      sim_wait_for( give_or_destroy_p,
event_received);
   }
   else { // check that deferred event is a GIVE or
DESTROY
      if ((event_received.type() != GIVE) &&
          (event_received.type() != DESTROY)){
         sim_error("Unexpected event type: %d received
in resource %s",
            event_received.type(), sim_name);
      }
   }
   return event_received;
}

sim_event
resources::next_take_back_or_destroy(sim_entity_id
from)
{
   const sim_type_p take_back_or_destroy_p( TAKE_BACK,
DESTROY);
   sim_event event_received;

   sim_wait_for( take_back_or_destroy_p,
event_received);
   if ( event_received.type() == TAKE_BACK ) {
      if ( event_received.scheduled_by() != from) {
         sim_error("Entity other than consumer has
given back resource %s \n",
         sim_name());
      }
  }
   return event_received;
}


void resources::body()
{
   sim_event next_event;
   sim_entity_id resource_requestor;

   next_event = next_give_or_destroy();
   while (next_event.type() == GIVE ) {
   // DESTROY results in termination

      resource_requestor = next_event.scheduled_by();
      // give the resource to the requestor
      sim_schedule( resource_requestor, 0.0,
GIVE_ACK);
      // wait for a request to give it back from the
requestor
      next_event =
next_take_back_or_destroy(resource_requestor);
      if (next_event.type() != DESTROY) {
         next_event = next_give_or_destroy();
      }
   }
}



//////test4.c
#include "/home/vtg/sim++/reusable/resources.h";
#include <stdlib.h>;

// message types
enum { INIT, GO, STOP};

// instantiate the class
SIM_ENTITY(resources);

// handle for the one instance
sim_entity_id resource_id;


// customer entity class
class customer : public sim_entity, public consumers {
     sim_time starting_delay;
   public:
      customer( sim_event &ev);
      void body();
};

customer::customer( sim_event &ev)
{
  // initialize the entity
  SIM_GET( sim_time, starting_delay, ev);
  }

void customer::body()
{
   sim_event ev;
```

```
   sim_hold_for( starting_delay, SIM_NONE, ev);
   acquire_resource( resource_id);
   sim_hold_for( 1000000.0, SIM_NONE, ev);
   give_back_resource( resource_id);

 }

// instantiate entity class
SIM_ENTITY(customer);


// create and initialize entities
void sim_initialize( int, char *argv[], int, char**)
{
     int n = atoi(argv[1]);
     char *res_name = "resource_entity";
     char *customer_name = "customer_entity";

     // create one instance of resource class with ID
resource_id and name
     // resource_name
     resource_id = sim_create("resources",res_name,
resources::INIT);

     // create n instances of customer class with name
customer_%i
     for ( int i = 1; i <= n; i++) {
        sprintf( customer_name, "customer_%d", i);
        sim_time start_delay = i;
        sim_create( "customer", customer_name, INIT,
            SIM_PUT( sim_time, start_delay));
    }
}
```

```
////// test5.c
#include <sim++.h>
#include <stdlib.h>

// message types
enum { INIT, INTERRUPT, STOP};

// id for entity instantiation
sim_entity_id foo_id;

// receiver entity class
class foo : public sim_entity {
   public:
     foo( sim_event &ev);
     void body();
     void tripper();
     void trippee();
   private:
     int trips;
};

foo::foo( sim_event &ev)
{
   SIM_GET( int, trips, ev);
 }

void foo::body()
{
   trippee();
 }

void foo::tripper() {
     sim_schedule( sim_current(), 1.0, INTERRUPT);
}

void foo::trippee()
{
    const sim_type_p interrupt( INTERRUPT);
    sim_event ev;
    while (0 < trips--)
    {
     // set the time out timer
     tripper();
     // wait for a long time (anticipating interrupt)
     sim_hold_for( 100.0, interrupt, ev);
    }
}

// instantiate entity class
SIM_ENTITY(foo);

// create and initialize entities
void sim_initialize( int, char *argv[], int, char*[])
{
     int n = atoi(argv[1]);
     char *foo_name = "foo_entity";

     // create one instance of class foo id ID foo_id
and name foo_name
     foo_id = sim_create( "foo", foo_name, INIT,
SIM_PUT( int, n));

   }
```

Sim++ BANK SIMULATION DESCRIPTION

The bank simulation in *Sim++* is the most complex of our benchmarks, and we include here some discuss before we list the source code. Figure 12 summarizes the event flows between entities in the simulation. Event flows are labelled with corresponding event identifiers. The termination of event flows are labelled with the method(s) invoked by the receiving entity's body method. The *summary* entity, the initialization event flows, and the report event flows are not shown in the figure. Not all methods are shown.

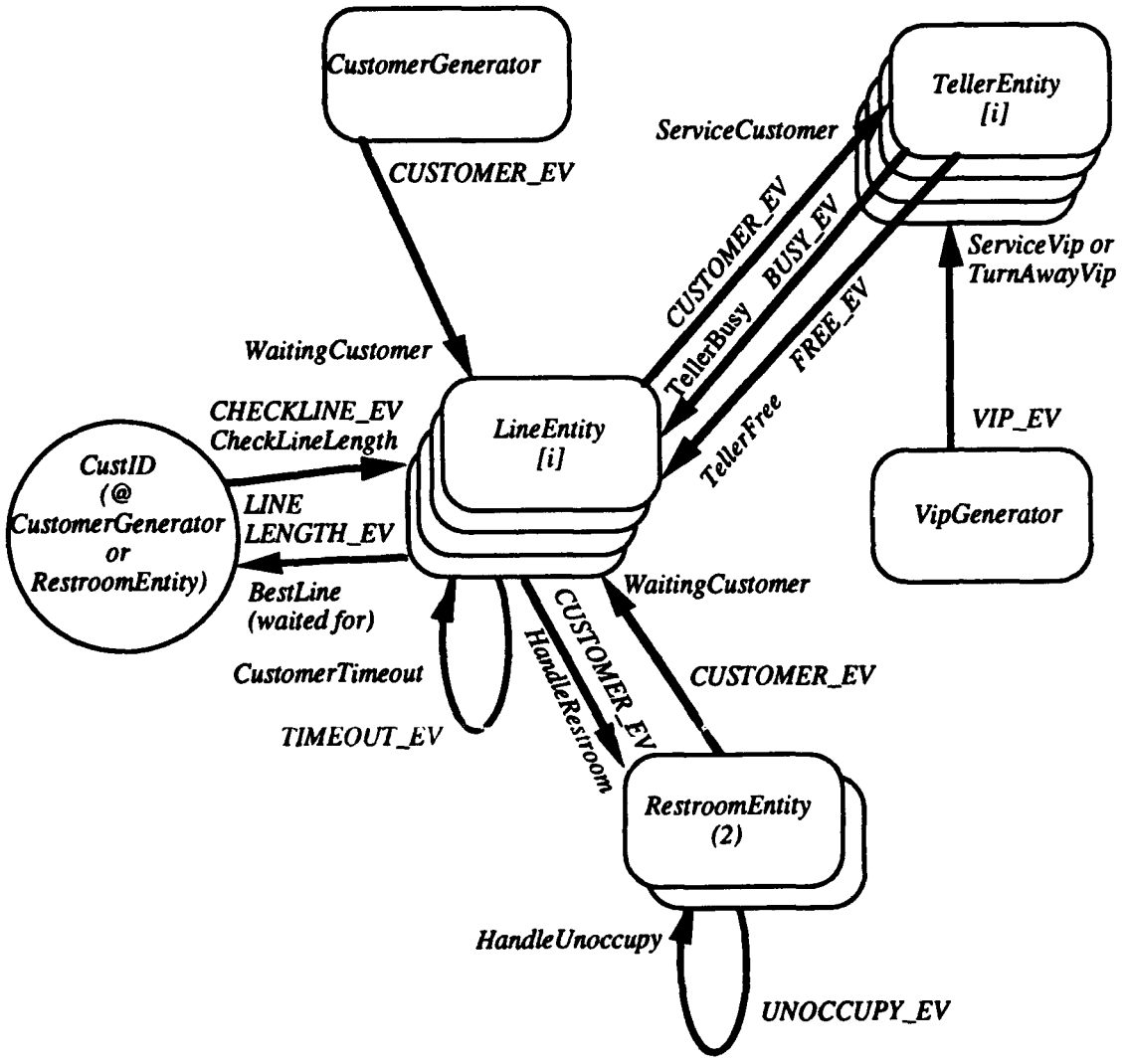


Figure 12. Major Entities and Event Flows

Six entity classes are used in the program:

1. *CustomerGenerator*: An instance of this class generated customers with a negative exponential interarrival time. The customer's gender and ID were assigned at this time. The customer is sent to the line determined by the customer's *BestLine* method.

2. *VipGenerator:* An instance of this class generated VIPs with a negative exponential interarrival time. The VIP's ID was assigned and the VIP was sent to a randomly-determined teller.

3. *LineEntity:* One instance of this class was associated with each teller. This entity class modeled the line in which the customers waited until a teller was busy or until their "restroom tolerance" was exceeded. In addition to the required constructor and body methods, the following additional methods were defined:

    a) *PendingNumber:* This method determined the line length.

    b) *CheckLineLength:* This method replied to a query for the length of a line.

    c) *WaitingCustomer:* This method handled the arrival of a customer on line. If the line were empty and the teller were free, the customer would be sent to the teller for service. Otherwise, the customer would be enqueued and the customer's *ArrivedOnLine* method would be invoked to schedule a timeout for its restroom break.

    d) *CustomerTimeout:* This method handled the occurrence of a restroom time for a customer. The customer was dequeued and sent to the restroom appropriate to the customer's gender.

    e) *TellerBusy:* This method handled an event that indicated that a line's teller was busy.

    f) *TellerFree:* Conversely, this method handled the event that indicated that a line's teller was free. If any customers were in line, the head of the line was removed, the customer's *LeavingLine* method was invoked in order to cancel the corresponding restroom timeout event, and the customer was sent to the line.

    g) *UpdateStats:* This method was invoked when the length of the line was changed so that the time-averaged line length could be accumulated.

    h) *WriteReport:* This method reported the accumulated line length statistics to the summary entity for consolidation and printing.

4. *TellerEntity:* Instances of this class were created to model the bank's tellers. In addition to the constructor and body methods, five methods were defined:

   a) *GetNextEvent:* This method informs the corresponding line that the teller is free and waits for the arrival of a customer or VIP.

   b) *ServiceCustomer:* This method determines the service time of a customer according to an Erlang distribution and attempts to service the customer for that time. This service may be interrupted by the arrival of a VIP, at which time the *ServiceVip* method is invoked. After return from this method, the teller attempts to give the customer the remaining service time that it needs. Service time data for the customers is accumulated.

   c) *ServiceVip:* This method determines the service time required by a VIP according to an Erlang distribution and attempts to service the VIP for that time. If interrupted by the arrival of another VIP, the *TurnAwayVip* method is invoked. After interruption, service is resumed. Service time statistics are accumulated.

   d) *TurnAwayVip:* This method refuses service to VIPs that attempt to interrupt the service of other VIPs. A count of the number of VIPs turned away is accumulated.

   e) *WriteReport*: This method reports the accumulated statistics to the summary entity.

5. *RestroomEntity:* An instance of the restroom entity class modeled the restroom for each gender. Each restroom is a single-queue multi-server. The methods defined in addition to the constructor and body methods are:

   a) *PendingNumber:* This method was reported the line length.

   b) *HandleRestroom:* This method handled the arrival of a customer. If there was no queue and a stall was free, the stall would be marked busy and the service time for the customer would be determined according to an Erlang distribution. A corresponding completion event would be scheduled. Otherwise, the line length would be checked against the customer's line length tolerance. If this tolerance was exceeded, the customer would exit the bank. A count of these customers was accumulated. If the tolerance were not exceeded, the customer would be enqueued.

   c) *HandleUnoccupy:* This method handled the completion event for a customer. The customer is sent to the line entity with the best line length as determined by the customer's *BestLine* method. The corresponding stall is marked free. If

there are customers enqueued, the head of the queue is removed, the stall is marked occupied, and a completion event is scheduled according the service time determined by an Erlang distribution.

d) *UpdateStats:* This method was called when the line length changed in order to accumulate the line length averaged over time.

e) *WriteReport*: This method reports the accumulated statistics to the summary entity.

6. *Summary:* One instance of this entity class was used to consolidate and print the statistical information for the simulation run.

Twelve event types were defined:

0. *INIT_EV.* This identifier was used for all "initialization events" sent to each entity during the *Sim++* initialization phase. "Init" object classes were defined for each of the entity classes that contained the data values necessary to identify or customize each entity instance. Objects of the appropriate "init" class were included in each event sent by the *sim_initialize* method to each created entity.

1. *CHECKLINE_EV.* This event requested that the receiving *line* entity report its line length to the sending *generator* entity. The body of these events were null.

2. *LINELENGTH_EV.* In response, a *line* entity would include an *int* value in an event with this identifier.

3. *CUSTOMER_EV.* Customers were sent between entities in events with this identifier. The body of these events contained an instance of the *CustID* class.

4. *TIMEOUT_EV.* When arriving on line, a customer would send an event with this identifier to the corresponding *line* entity that would arrive at the entity if the customer's "restroom tolerance" were exceeded. The body of these events contained a copy of the *CustID* generating the event.

5. *VIP_EV.* The arrival of a VIP was indicated by the receipt of an event with this identifier. The body of the event would contain an object of the *VipID* class.
6. *UNOCCUPY_EV.* The *RestroomEntity* entities scheduled events with this identifier to indicate the time when service for a customer would be completed. The bodies of these events contained the customer objects.

7. *BUSY_EV.* The *TellerEntity* entities sent events with this identifier to their corresponding *LineEntity* entities to indicate that they were busy servicing a customer

or a VIP. This event was necessary because the arrival of VIP at an otherwise free teller should block the normal customer from arriving. The body of these events were empty.

8. *FREE_EV.* Conversely, this event indicates to a *LineEntity* that the corresponding *TellerEntity* is free and that the next customer, when available, should be sent to the teller.

9. *REPORT_EV.* Events with this identifier were used by the *Summary* entity to request statistics from the other entities.

10. *REPORT_REPLY.* The responses were returned to the *Summary* entity in events with this identifier. These responses contained objects of classes associated with the entity responding.

11. *LAST_EV.* An event with this identifier was sent by each generator to indicate that the simulation end time was exceeded and that the summary statistics should be collected and printed.

Several C++ classes were developed that were not *Sim++* entity classes:

1. *CustID:* This class carried that data associated with each customer. It was carried as the body in several types of events. Other than a constructor, it has methods:

   a) *BestLine:* This method determines the line with the shortest length.

   b) *Arrived_On_Line:* This method determines the customer's restroom tolerance and schedules a *TIMEOUT_EV* event.

   c) *Leaving_Line:* This method cancels the *TIMEOUT_EV* event scheduled when the customer arrived.

   d) *Service_Started:* This method saves the start of service by the teller for later statistics.

2. *VipID:* This class carries the data associated with the VIPs and was carried in the body of events with the *VIP_EV* identifier. The constructor was the only method defined.
3. *my_tally:* This class was a modification of the *Sim++*-provided class *tally*. It added an *operator+=* method that efficiently consolidates the statistics gathered by two instances.

Sim++ BANK SIMULATION CODE

```
////// bank simulation


////// bank.h
#include <sim++.h>
#ifndef BANK_H
#define BANK_H

const MAX_LINE_LEN = 10000;
const MAX_LINES = 20;
const MAXNUM    = 20;
const LEVEL1    = 0;
const MIN_ONQ   = 100;

enum Gender (Female, Male);
enum TellerState ( Free, Busy);

enum (   INIT_EV,
     CHECKLINE_EV,
     LINELENGTH_EV,
     CUSTOMER_EV,
     TIMEOUT_EV,
     VIP_EV,
     UNOCCUPY_EV,
     BUSY_EV,
     FREE_EV,
     REPORT_EV,
     REPORT_REPLY,
        LAST_EV );


// global parameters

extern double   hours_to_run;
extern int      numlines;
extern int      numrests;

extern sim_entity_id line_id[MAX_LINES+1];
extern sim_entity_id rest_id[2];
extern sim_entity_id teller_id[MAX_LINES+1];
extern sim_entity_id cust_g_id;
extern sim_entity_id vip_g_id;
extern sim_entity_id summary_id;

int line_length[MAX_LINES+1];
#endif


////// custid.h
#ifndef CUSTID_H
#define CUSTID_H
#include "bank.h"


// Customer id. object class
class CustID
{
  sim_event_id timeout_event;
 public:
  Gender gender;
  int line, id, toilet;
  sim_time on_queue, service_start;
  CustID() {line = 0; id = 0; toilet = 0; on_queue =
0; service_start = 0;}
  int BestLine();
  void Arrived_on_Line(sim_erlang_obj
*timeout_generator);
  void Leaving_Line();
  void Service_Started();
};

SIM_DECLARE_LIST(CustID);


class VipID
{
 public:
  int line;
  int id;
  VipID() { line = 0; id = 0; }
};

#endif


///// generator.h
#include "bank.h"


// CustomerSource object declaration
class CustomerGeneratorInit
{
  public:
    int gender_seed;
    double arrival_rate;
    int arrival_seed;
};

class CustomerGenerator : public sim_entity

    sim_randint_obj gender_generator;
    sim_negexp_obj  interarrival_time_generator;
  public:
    CustomerGenerator(sim_event &ev);
    void body();
};
SIM_ENTITY(CustomerGenerator);


// VipSource object declaration
class VipGeneratorInit
{
  public:
    int line_seed;
    double arrival_rate;
    int arrival_seed;
};

class VipGenerator : public sim_entity
{
    sim_randint_obj line_generator;
    sim_negexp_obj interarrival_time_generator;
    int vip_generated;
  public:
    VipGenerator(sim_event &ev);
    void body();
};
SIM_ENTITY(VipGenerator);


////// line.h
#include "bank.h"
#include "custid.h"
#include "my_tally.h"

// Line object initialization parameters
class LineInit
{
  public:
    int ID;
    sim_erlang_obj *timeout_generator;
};

// Line object stats class
class LineStats
{
 public:
  sim_accum cust_online;
  double last_event_time;
  my_tally cust_waiting_time;
};
```

```
// Line object entity declaration
class LineEntity : public sim_entity
{
  int ID;
  TellerState status;
  CustID_head *queue;
  LineStats stats;
  sim_erlang_obj *timeout_generator;
 public:
  LineEntity(sim_event &ev);
  void body();
  int PendingNumber();
  void CheckLineLength(sim_entity_id &requester);
  void WaitingCustomer(CustID &customer);
  void CustomerTimeout(CustID &customer);
  void TellerBusy();
  void TellerFree();
  void UpdateStats();
  void WriteReport();
};
SIM_ENTITY(LineEntity);
```

```
//////// my_tally.h
#include <sim++.h>
#ifndef MY_TALLY_H
#define MY_TALLY_H

class my_tally : public sim_tabulate {
  double Sum, Sumsq, Min, Max;
public:
  my_tally();
  my_tally(const char *title);
  void reset();
  void update(double v);
  void my_tally::operator+=(my_tally tally);
  int obs() const { return sim_tabulate::obs(); }
  double avg() const;
  double std_dev() const;
  double min() const;
  double max() const;
  const char *title() const { return
sim_tabulate::title(); }
  void report() const;
  void freport( const sim_file_id &file) const;
};

#endif
```

```
//////// restroom.h
#include "bank.h"
#include "custid.h"

//Restroom initialization object
class RestroomInit
{
  public:
    Gender         gender;
    sim_erlang_obj *restroom_service_generator;
    sim_erlang_obj *restroom_tolerance_generator;
};

// Restroom stats object
class RestroomStats
{
 public:
  int customers, cust_walkaway;
  sim_accum line_length;
  sim_time last_event_time;
};

// Restroom object entity declaration
class RestroomEntity : public sim_entity
{
  Gender gender;
  CustID_head *cust_list;
  int occupy[MAX_LINES+1];
  sim_erlang_obj *service_generator;
  sim_erlang_obj *tolerance_generator;
  RestroomStats stats;
 public:
  RestroomEntity(sim_event &ev);
  void body();
  int PendingNumber();
  void HandleRestroom(sim_event &ev);
  void HandleUnoccupy(sim_event &ev);
  void UpdateStats();
  void WriteReport();
};
SIM_ENTITY(RestroomEntity);
```

```
////// summary.h
#include "bank.h"

// Summary object declaration
class Summary : public sim_entity
{
  int total_report;
 public:
  Summary(sim_event&);
  void body();
};
SIM_ENTITY(Summary);
```

```
//////// teller.h
#include "bank.h"
#include "custid.h"
#include "my_tally.h"

// Teller Initialization parameters
class TellerInit
{
  public:
    int ID;
    sim_erlang_obj *customer_service_generator;
    sim_erlang_obj *vip_service_generator;
};

// Teller Stats object
class TellerStats
{
 public:
  int cust_interrupts;
  int vip_turnaway;
  my_tally customer_service_time;
  my_tally VIP_service_time;
  TellerStats() { cust_interrupts = 0; vip_turnaway =
0;}
};

// Teller object entity declaration
class TellerEntity : public sim_entity
{
  int ID;
  sim_erlang_obj *customer_service_generator;
  sim_erlang_obj *vip_service_generator;
  TellerStats stats;
 public:
  TellerEntity(sim_event &ev);
  void body();
  void GetNextEvent(sim_event &ev);
  void ServiceCustomer(CustID customer);
  void ServiceVip(VipID VIP);
  void TurnAwayVip(VipID VIP);
  void WriteReport();
};
SIM_ENTITY(TellerEntity);
```

```
////// bank.c
#include <stdlib.h>
#include "bank.h"
#include "generator.h"
#include "line.h"
#include "restroom.h"
#include "teller.h"
```

```
// configuration parameters of the bank
sim_time hours_to_run;
int      numlines;
int      numrests;

// entity id for all objects
sim_entity_id line_id[MAX_LINES+1];
sim_entity_id rest_id[2];
sim_entity_id teller_id[MAX_LINES+1];
sim_entity_id cust_g_id;
sim_entity_id vip_g_id;
sim_entity_id summary_id;

// the common random number generators
sim_erlang_obj restroom_timeout_generator;
sim_erlang_obj restroom_service_generator[2];
sim_erlang_obj restroom_tolerance_generator; //shared
by both genders
sim_erlang_obj customer_service_generator;
sim_erlang_obj vip_service_generator;

  double          cust_arrv_rate;
  int             gcust_seed;
  double          vip_arrv_rate;
  int             gvip_seed;
  int             gline_seed;
  int             ggender_seed;

void read_data_file(char* datav[])
{
  double          tc_mean;
  int             tc_sample;
  int             gtc_seed;
  double          rt_mean;
  int             rt_sample;
  int             grt_seed;
  double          rr_mean[2];
  int             rr_sample[2];
  int             grr_seed[2];
  double          cust_service_mean;
  int             cust_service_sample;
  int             gcust_service_seed;
  double          vip_service_mean;
  int             vip_service_sample;
  int             gvip_service_seed;

  // the random number parameters read from the inputs
  // customer and vip arrival rates and their seeds
  sscanf(datav[0], "%lg %d %lg %d", &cust_arrv_rate,
&gcust_seed,
      &vip_arrv_rate, &gvip_seed);

  // customer and vip service times (mean, sample and
seed)
  sscanf(datav[1], "%lg %d %d %lg %d %d",
&cust_service_mean,
        &cust_service_sample, &gcust_service_seed,
&vip_service_mean,
        &vip_service_sample, &gvip_service_seed);
  customer_service_generator =
sim_erlang_obj("Customer Service Time",
    cust_service_mean, cust_service_sample,
gcust_service_seed);
  vip_service_generator = sim_erlang_obj("VIP Service
Time",
     vip_service_mean, vip_service_sample,
gvip_service_seed);

  // customer's time spent in restroom (mean, sample
and seed)
  sscanf(datav[2], "%lg %d %d %lg %d %d", &rr_mean[0],
      &rr_sample[0], &grr_seed[0], &rr_mean[1],
      &rr_sample[1], &grr_seed[1]);
  restroom_service_generator[0] =
sim_erlang_obj("Female Service",
      rr_mean[0], rr_sample[0], grr_seed[0]);
  restroom_service_generator[1] = sim_erlang_obj("Male
Service",
      rr_mean[1], rr_sample[1], grr_seed[1]);

  // customer's waiting time on a line before leaving
for restroom
  // and restroom tolerance
  sscanf(datav[3], "%lg %d %d %lg %d %d", &tc_mean,
&tc_sample, &gtc_seed,
      &rt_mean, &rt_sample, &grt_seed);
  restroom_timeout_generator =
sim_erlang_obj("Restroom timeout",
     tc_mean, tc_sample, gtc_seed);
  restroom_tolerance_generator =
sim_erlang_obj("Female tolerance",
     rt_mean, rt_sample, grt_seed);

  // line seed and gender seed for their randint
function
  sscanf(datav[4], "%d %d", &gline_seed,
&ggender_seed);

  // number of lines in the bank, number of toilers in
each restroom
  // and number of hour to run the simulation
  sscanf(datav[5], "%d %d %lg", &numlines, &numrests,
&hours_to_run);

  sim_trace(LEVEL1, "initalizing all input
parameters\n");

}


void sim_initialize(int, char *[], int, char* datav[])
{
  char line_name[20], teller_name[20], name[20];

  read_data_file(datav);

  sim_trace(LEVEL1, "creating line and restroom
entities\n");

  for (int i=1; i<=::numlines; i++) {

    sprintf(line_name, "line_entity_%d", i);
    LineInit line_init_info;
    line_init_info.ID = i;
    line_init_info.timeout_generator =
&restroom_timeout_generator;
    ::line_id[i] = sim_create("LineEntity", line_name,
INIT_EV,
                        SIM_PUT(LineInit,
line_init_info));

    sprintf(teller_name, "teller_entity_%d", i);
    TellerInit teller_init_info;
    teller_init_info.ID = I;
    teller_init_info.customer_service_generator =
&customer_service_generator;
    teller_init_info.vip_service_generator =
&vip_service_generator;
    ::teller_id[i] = sim_create("TellerEntity",
teller_name, INIT_EV,
                          SIM_PUT(TellerInit,
teller_init_info));
  }

  for (Gender j = Female; j <= Male; j++) {
    sprintf(name, "restroom_entity_%d", j);
    RestroomInit restroom_init_info;
    restroom_init_info.gender = j;
    restroom_init_info.restroom_service_generator =
     &restroom_service_generator[j];
    restroom_init_info.restroom_tolerance_generator =
     &restroom_tolerance_generator;
    ::rest_id[j] = sim_create("RestroomEntity", name,
INIT_EV,
                        SIM_PUT(RestroomInit,
restroom_init_info));
  }

  CustomerGeneratorInit custgen_init;
  custgen_init.gender_seed = ggender_seed;
  custgen_init.arrival_rate = cust_arrv_rate;
  custgen_init.arrival_seed = gcust_seed;
```

```
  cust_g_id = sim_create("CustomerGenerator",
"customer_generator", INIT_EV,
                SIM_PUT(CustomerGeneratorInit,
custgen_init));

  VipGeneratorInit  vipgen_init;
  vipgen_init.line_seed = gline_seed;
  vipgen_init.arrival_rate = vip_arrv_rate;
  vipgen_init.arrival_seed = gvip_seed;
  vip_g_id = sim_create("VipGenerator",
"vip_generator", INIT_EV,
                SIM_PUT(VipGeneratorInit,
vipgen_init));

  summary_id = sim_create("Summary", "summary",
INIT_EV);
}



///////// custid.c
#include "custid.h"

int CustID::BestLine()
{
  int bestline;
  int bestlen = MAX_LINE_LEN;

  for(int line = 1; line <=:: numlines; line++) {
    if (line_length[line] == 0) {
      bestline = line;
      break;
    }
    if (line_length[line] < bestlen){
      bestline = line;
      bestlen = line_length[line];
    }
  }
  return bestline;
}

void CustID::Arrived_on_Line(sim_erlang_obj
*timeout_generator)
{
#   ifdef INSTRUMENTED
    if ( on_queue == 0.0)
     on_queue = sim_clock();
#   endif

    double timeout = timeout_generator->sample();
    timeout_event = sim_schedule(sim_current(),
timeout, TIMEOUT_EV,
                             SIM_PUT(CustID, *this));
}

void CustID::Leaving_Line()
{
    sim_cancel(timeout_event);
}

void CustID::Service_Started()
{
#   ifdef INSTRUMENTED
    if ( service_start == 0.0)
     service_start = sim_clock();
#   endif
}


//////// generator.c
#include "generator.h"
#include "custid.h"

/*
** CustomerGenerator object entity implementation
*/

CustomerGenerator::CustomerGenerator(sim_event &ev)
{
    sim_trace(LEVEL1, "Initializing customer
generator\n");
    CustomerGeneratorInit init_info;
    SIM_GET(CustomerGeneratorInit, init_info, ev);
    gender_generator = sim_randint_obj("Customer
Gender", 0, 1,

init_info.gender_seed);
    Interarrival_time_generator =
sim_negexp_obj("Customer Interarrival",
     init_info.arrival_rate, init_info.arrival_seed);
}

void CustomerGenerator::body()
{
  int total_cust = 0;
  sim_event ev;
  sim_time interarrival_time;
  CustID tag;

  while (sim_clock() <= ::hours_to_run*60) {
    total_cust ++;
    sim_trace(LEVEL1, "Generating customer #: %d\n",
total_cust);
    tag.gender = (Gender) gender_generator.sample();
    tag.line = tag.BestLine();
    tag.id = total_cust;
    sim_schedule(::line_id[tag.line], 0.0,
CUSTOMER_EV, SIM_PUT(CustID, tag));

    interarrival_time =
interarrival_time_generator.sample();
    sim_hold_for(interarrival_time, SIM_NONE, ev);
   }

  //indicate to summary that no more customers will
arrive
  sim_schedule(::summary_id, 0.0, LAST_EV,
SIM_PUT(int,total_cust));
}


/*
** VipCustomerGenerator object implementation
*/

VipGenerator::VipGenerator(sim_event &ev)
{
    sim_trace(LEVEL1, "entering vip constructor\n");
    VipGeneratorInit init_info;
    SIM_GET(VipGeneratorInit, init_info, ev);
    line_generator = sim_randint_obj("VIP Line", 1,
numlines,

init_info.line_seed);
    Interarrival_time_generator = sim_negexp_obj("VIP
Interarrival",
     init_info.arrival_rate, init_info.arrival_seed);
}

void VipGenerator::body()
{
  int total_vip = 0;
  VipID vip;
  sim_event ev;
  sim_time interarrival_time;

  interarrival_time =
interarrival_time_generator.sample();
  sim_hold_for(interarrival_time, SIM_NONE, ev);
  while (sim_clock() <  ::hours_to_run*60) {
    total_vip++;
    sim_trace(LEVEL1, "Generating VIP customer
#%d\n",total_vip);
    vip.line = line_generator.sample();
    vip.id = total_vip;
    sim_schedule(::teller_id[vip.line], 0.0, VIP_EV,
SIM_PUT(VipID, vip));

    interarrival_time =
interarrival_time_generator.sample();
    sim_hold_for(interarrival_time, SIM_NONE, ev);
  }
```

```
  //indicate to summary that no more customers will
arrive
  sim_schedule(::summary_id, 0.0, LAST_EV,
SIM_PUT(int,total_vip));
}



//////// line.c
#include "line.h"
#include <stdio.h>
/*
** Line object entity implementation
*/

LineEntity::LineEntity(sim_event &ev)
{
    char title[20];

    sim_trace(LEVEL1, "Line entity initializing\n");
    LineInit init_info;
    SIM_GET(LineInit, init_info, ev);
    ID = init_info.ID;
    timeout_generator = init_info.timeout_generator;

    status = Free;
    queue = new CustID_head("queue");
    line_length[ID] = 0;

#   ifdef INSTRUMENTED
    sprintf(title,"Line %d",ID);
    stats.cust_online = sim_accum(title);
    stats.last_event_time = 0;
#   endif
}

void LineEntity::body()
{
  sim_event ev;
  CustID customer;

  sim_wait(ev);
  while (ev.type() != REPORT_EV) {
    switch(ev.type()) {
      case CHECKLINE_EV:
        CheckLineLength(ev.scheduled_by());
    break;
      case CUSTOMER_EV:
    SIM_GET(CustID, customer, ev);
    WaitingCustomer(customer);
    break;
      case TIMEOUT_EV:
    SIM_GET(CustID, customer, ev);
    CustomerTimeout(customer);
    break;
      case BUSY_EV:
    TellerBusy();
    break;
      case FREE_EV:
    TellerFree();
    break;
      default:
        sim_error("unexpected event type %d from:
%s\n", ev.type(),
            ev.scheduled_by().name());
      }
    sim_select(SIM_ANY, ev);
    if (ev == SIM_NO_EVENT) {
      sim_trace(LEVEL1, "Waiting for next customer or
vip\n");
      sim_wait(ev);
    }
  }
# ifdef INSTRUMENTED
  WriteReport();
# endif

  sim_trace(LEVEL1, "Line Terminated\n");
}

int LineEntity::PendingNumber()
{
  if ((!queue->empty()) && (status == Free))
    return (queue->cardinal());
  else if ((!queue->empty()) && (status == Busy))
    return (queue->cardinal() +1);
  else if (status == Busy)
    return 1;
  else
    return 0;
}

void LineEntity::UpdateStats()
{
#   ifdef INSTRUMENTED
    stats.cust_online.update( (sim_clock() -
stats.last_event_time),
              double(PendingNumber()));
    stats.last_event_time = sim_clock();
#   endif
}

void LineEntity::CheckLineLength(sim_entity_id
&requester)
{
  sim_trace(LEVEL1, "handling check line event\n");
  int length = PendingNumber();
  sim_schedule(requester, 0.0, LINELENGTH_EV,
SIM_PUT(int, length));
}

void LineEntity::WaitingCustomer(CustID &customer)
{
  CustID_elem *elem;

  sim_trace(LEVEL1, "handling waiting event\n");

  // data collection, e.g. ave. queue length
# ifdef INSTRUMENTED
  UpdateStats();
# endif

  // if the line is empty and the teller is free
  if ((queue->empty() > 0) && (status == Free)) {
//send directly to teller
#   ifdef INSTRUMENTED
    stats.cust_waiting_time.update(0.0);
#   endif
    sim_schedule(::teller_id[ID], 0.0, CUSTOMER_EV,
SIM_PUT(CustID, customer));
  }
  else
  {
    customer.Arrived_on_Line(timeout_generator);  //
schedule restroom
    // enqueue the customer on the line
    elem = new CustID_elem(customer);
    elem->append(queue);
    line_length[ID] ++;
  }
}

void LineEntity::CustomerTimeout(CustID &customer)
//Customer may have been on line too long, needs a
trip to the restroom
{
  CustID temp;
  CustID_elem *elem;

  sim_trace(LEVEL1, "handling timeout event\n");

  //must search for customer in queue
  elem = queue->first();
  while (elem != 0)
  {
      temp = elem->contents();
      if (temp.id == customer.id) {  //found customer
in line
        //take off line

#   ifdef INSTRUMENTED
        UpdateStats();
#   endif
    elem->out();
```

```
    //send to restroom
    int j = (int) customer.gender;
    sim_schedule(::rest_id[j], 0.0, CUSTOMER_EV,
            SIM_PUT(CustID, customer));
    delete elem;
    line_length[ID] --;
    return;
      }
      else
       elem = elem->next();
  }
  sim_error("Couldn't find timed out customer in
line\n");
}


void LineEntity::TellerBusy()
{

  sim_trace(LEVEL1, "handling busy event\n");
  status = Busy;
}

void LineEntity::TellerFree()
{
  CustID_elem *elem;
  CustID nextCustomer;

  sim_trace(LEVEL1, "handling free event\n");
  status = Free;
  if (!queue->empty()){
#   ifdef INSTRUMENTED
    UpdateStats();
#   endif
    elem = queue->first();
    elem->out();
    nextCustomer = elem->contents();
    nextCustomer.Leaving_Line();    // cancel restroom
trip
    line_length[ID]--;
#   ifdef INSTRUMENTED
    stats.cust_waiting_time.update( (sim_clock() -
nextCustomer.on_queue));
#   endif
    sim_schedule(::teller_id[ID], 0.0, CUSTOMER_EV,
          SIM_PUT(CustID, nextCustomer));
    delete elem;
  }
}


void LineEntity::WriteReport()
{
  sim_schedule(summary_id, 0.0, REPORT_REPLY,
SIM_PUT(LineStats,stats));
}



//////// my_tally.c
#include "my_tally.h"
#include <math.h>

my_tally::my_tally(): ("my_tally")
{
    Sum = 0.0;
    Sumsq = 0.0;
    Min = 1.0e55;  // very large
    Max = -1.0e55; // very large negative
}

my_tally::my_tally(const char *title) : (title,
"my_tally")
{
    Sum = 0.0;
    Sumsq = 0.0;
    Min = 1.0e55;  // very large
    Max = -1.0e55; // very large negative
}

void my_tally::reset()
{
    sim_tabulate::reset();
    Sum = 0.0;
    Sumsq = 0.0;
    Min = 1.0e55;  // very large
    Max = -1.0e55; // very large negative
}

void my_tally::update(double v)
{
    sim_tabulate::update_obs();
    Sum += v;
    Sumsq += (v*v);
    if ( v < Min ) Min = v;
    if ( v > Max ) Max = v;
}

void my_tally::operator+=(my_tally tally)
{
    update_obs(tally.obs());
    Sum += tally.Sum;
    Sumsq += tally.Sumsq;
    if ( tally.Max > Max) Max = tally.Max;
    if ( tally.Min < Min) Min = tally.Min;
}

double my_tally::avg() const
{
    double obs = double( sim_tabulate::obs());
    return Sum / obs;
}

double my_tally::std_dev() const
{
    double obs = double( sim_tabulate::obs());
    return sqrt(( (Sumsq - (Sum * Sum) / obs )/ (obs -
1.0) ));
}

double my_tally::min() const { return Min; }

double my_tally::max() const { return Max; }

void my_tally::report() const
{
    sim_printf(
            "%12s %6d %9.3f %9.3f %9.3f %9.3f %9.3f
\n",
            title(),
            obs(),
            Sum,
            avg(),
            std_dev(),
            min(),
            max()
            );
}

void my_tally::freport( const sim_file_id &file) const
{
    sim_fprintf(
            file,
            "%12s %6d %9.3f %9.3f %9.3f %9.3f %9.3f
\n",
            title(),
            obs(),
            Sum,
            avg(),
            std_dev(),
            min(),
            max()
            );
}



//////// restroom.c
#include "restroom.h"
#include <math.h>

RestroomEntity::RestroomEntity(sim_event &ev)
{
```

```
  sim_trace(LEVEL1, "Restroom Constructor\n");

  RestroomInit init_info;
  SIM_GET(RestroomInit, init_info, ev);
  gender = init_info.gender;
  service_generator =
init_info.restroom_service_generator;
  tolerance_generator =
init_info.restroom_tolerance_generator;

  for(int i=1; i<=::numrests; i++)
    occupy[i] = 0;
  cust_list = new CustID_head("cust_list");

# ifdef INSTRUMENTED
  stats.customers = 0;
  stats.cust_walkaway = 0;
  if (gender == Female)
      stats.line_length = sim_accum("Female");
  else
      stats.line_length = sim_accum("Male");
  stats.last_event_time = 0;
# endif
}

void RestroomEntity::body()
{
  sim_event ev;
  CustID tag;

  sim_wait(ev);
  while (ev.type() != REPORT_EV) {
    switch(ev.type()) {
      case CUSTOMER_EV:
        HandleRestroom(ev);
     break;
      case UNOCCUPY_EV:
     HandleUnoccupy(ev);
     break;
      default:
     sim_error("unexpected event type %d from %s\n",
ev.type(),
          ev.scheduled_by().name());
      }
    sim_select(SIM_ANY, ev);
    if (ev == SIM_NO_EVENT) {
      sim_trace(LEVEL1, "Waiting for next customer or
vip\n");
      sim_wait(ev);
    }
  }
# ifdef INSTRUMENTED
  WriteReport();
# endif

  sim_trace(LEVEL1, "Restroom Terminated\n");
}

void RestroomEntity::HandleRestroom(sim_event &ev)
{
  CustID tag;
  double restroom_duration;
  int restroom_tolerance;
  CustID_elem *elem;


  SIM_GET(CustID, tag, ev);

# ifdef INSTRUMENTED
  stats.customers++;
# endif

  for (int i=1; i <= ::numrests; i++)
    if (occupy[i] == 0)
      break;

  if ((cust_list->empty() > 0) && (i <= ::numrests)) {
    sim_trace(LEVEL1, "restroom is available\n");
    occupy[i] = 1;
    tag.toilet = i;
    restroom_duration = service_generator->sample();
    sim_schedule(sim_current(), restroom_duration,
UNOCCUPY_EV,
          SIM_PUT(CustID, tag));
  }
  else {
    sim_trace(LEVEL1, "there is a line or all toilets
are occupied\n");
    restroom_tolerance =(int) anint(
tolerance_generator->sample() );
    if (PendingNumber() > restroom_tolerance) {
#     ifdef INSTRUMENTED
      stats.cust_walkaway++; // customer walk out of
bank
#     endif
      return;
    }
    else {
      // enqueue it on the list
      elem = new CustID_elem(tag);
      elem->append(cust_list);
#     ifdef INSTRUMENTED
      UpdateStats();
#     endif
    }
  }
}

int RestroomEntity::PendingNumber()
{
  return (cust_list->cardinal());
}

void RestroomEntity::HandleUnoccupy(sim_event &ev)
{
  CustID_elem *elem;
  CustID tag, temp;
  double restroom_duration;

  SIM_GET(CustID, tag, ev);
  tag.line = tag.BestLine();
  sim_schedule(::line_id[tag.line], 0.0, CUSTOMER_EV,
SIM_PUT(CustID, tag));
  int i = tag.toilet;
  occupy[i] = 0;
  if (!cust_list->empty()) {
    occupy[i] = 1;
    elem = cust_list->first();
    elem->out();
    temp = elem->contents();
    delete elem;
#   ifdef INSTRUMENTED
    UpdateStats();
#   endif
    restroom_duration = service_generator->sample();
    sim_schedule(sim_current(), restroom_duration,
UNOCCUPY_EV,
          SIM_PUT(CustID, temp));
  }
}

void RestroomEntity::UpdateStats()
{
      int num_onQ = PendingNumber();
      stats.line_length.update(
        (sim_clock() - stats.last_event_time),
double(num_onQ));
      stats.last_event_time = sim_clock();
}

void RestroomEntity::WriteReport()
{
  sim_trace(LEVEL1, "Writing Restroom Report: Restroom
%d \n", gender);
  sim_schedule(summary_id, 0.0, REPORT_REPLY,
SIM_PUT(RestroomStats,stats));
}



//////// summary.c
#include "summary.h"
#include "line.h"
```

```
#include "teller.h"
#include "restroom.h"

Summary::Summary(sim_event&)
{
  sim_trace(LEVEL1, "summary constructor\n");
  total_report=0;
}

void Summary::body()
{
  sim_event ev;
  sim_from_p customer_generator( ::cust_g_id);
  sim_from_p vip_generator( ::vip_g_id);
  sim_type_p report_reply_p(REPORT_REPLY);
# ifdef INSTRUMENTED
  int number;
  sim_file_id report_file;
  LineStats line_length_stats;
  TellerStats teller_totals;
  TellerStats teller_report;
  RestroomStats restroom_report;

  // generate the report
  report_file = sim_fopen("report","w");
# endif

//Generators
  //wait for both generators to finish
  sim_wait_for( customer_generator, ev);
# ifdef INSTRUMENTED
  SIM_GET(int, number, ev);
  sim_fprintf(report_file, "%d customers arrived\n",
number);
# endif
  sim_wait_for( vip_generator, ev);
# ifdef INSTRUMENTED
  SIM_GET(int, number, ev);
  sim_fprintf(report_file, "%d VIPS arrived\n",
number);
# endif

//Lines
  //print the statistics for the line lengths
# ifdef INSTRUMENTED
  my_tally customer_waiting_time("Waiting Time");
  sim_fprintf(report_file,
        "-----------------------------------------
---------------\n");
  sim_fprintf(report_file, "Line Statistics\nLine
Lengths\n");
  sim_fprintf(report_file, ::sim_accum_heading());
# endif
  for (int i = 1; i <= ::numlines; i++) {
    sim_schedule(::line_id[i], 0.0, REPORT_EV);
#   ifdef INSTRUMENTED
    sim_wait_for( report_reply_p, ev);
    SIM_GET(LineStats, line_length_stats, ev);

line_length_stats.cust_online.freport(report_file);
    customer_waiting_time +=
line_length_stats.cust_waiting_time;
#   endif
  }
# ifdef INSTRUMENTED
  sim_fprintf(report_file, ::sim_tally_heading());
  customer_waiting_time.freport(report_file);
# endif

//Restrooms
# ifdef INSTRUMENTED
  int total_customers = 0;
  int total_customers_walking_away = 0;
  sim_fprintf(report_file,
        "-----------------------------------------
---------------\n");
  sim_fprintf(report_file, "Restroom Statistics\nLine
Lengths\n");
  sim_fprintf(report_file, ::sim_accum_heading());
# endif
  for (i = 0; i<=1; i++) {
    sim_schedule(::rest_id[i], 0.0, REPORT_EV);
#   ifdef INSTRUMENTED
    sim_wait_for( report_reply_p, ev);
    SIM_GET(RestroomStats, restroom_report, ev);
    restroom_report.line_length.freport(report_file);
    total_customers += restroom_report.customers;
    total_customers_walking_away +=
restroom_report.cust_walkaway;
#   endif
  }
# ifdef INSTRUMENTED
  sim_fprintf(report_file, "There were %5d trips to
the rest rooms\n",
          total_customers);
  sim_fprintf(report_file, "%5d customers walked away
without service\n",
          total_customers_walking_away);
# endif

//Tellers
  //sum up the totals for the tellers
# ifdef INSTRUMENTED
  teller_totals.customer_service_time = my_tally("All
Customers");
  teller_totals.VIP_service_time = my_tally("All
VIPs");
  teller_totals.cust_interrupts = 0;
  teller_totals.vip_turnaway = 0;
  sim_fprintf(report_file,
        "-----------------------------------------
---------------\n");
  sim_fprintf(report_file, "Teller Service
Reports\n");
  sim_fprintf(report_file, ::sim_tally_heading());
# endif
  for (i = 1; i <= ::numlines; i++) {
    sim_schedule(::teller_id[i], 0.0, REPORT_EV);
#   ifdef INSTRUMENTED
    sim_wait_for( report_reply_p, ev);
    SIM_GET(TellerStats, teller_report, ev);

teller_report.customer_service_time.freport(report_fil
e);
    teller_totals.customer_service_time +=
        teller_report.customer_service_time;

teller_report.VIP_service_time.freport(report_file);
    teller_totals.VIP_service_time +=
        teller_report.VIP_service_time;
    teller_totals.cust_interrupts +=
teller_report.cust_interrupts;
    teller_totals.vip_turnaway +=
teller_report.vip_turnaway;
#   endif
  }
# ifdef INSTRUMENTED

teller_totals.customer_service_time.freport(report_fil
e);
  teller_totals.VIP_service_time.freport(report_file);
  sim_fprintf(report_file,
          "Number of interrupts by VIPs: %d\n",
          teller_totals.cust_interrupts);
  sim_fprintf(report_file,
          "Number of vips turned away without being
serviced: %d\n",
          teller_totals.vip_turnaway);

  sim_fprintf(report_file,
        "-----------------------------------------
---------------\n");
  //sim_fprintf(report_file, "total_cust = %d \t
total_vip = %d\n",
  //          total_cust, total_vip);
  //sim_fprintf(report_file,
"ave_waiting_on_queue_time = %f\n",
  //     total_queued_time/total_queued_cust);
  //sim_fprintf(report_file, "number of customers on
lines: %d\n",
  //          sum_line_length_stats.cust_online);
  //sim_fprintf(report_file, "number of customers
```

```
visiting restroom: %d\n",
  //
sum_line_length_stats.cust_timeout);
  //sim_fprintf(report_file,
        //"number of customers left without being
serviced: %d\n",
            //sum_restroom_report.cust_walkaway);
# endif
}



///////// teller.c
#include "teller.h"
#include <stdio.h>

/*
** Teller object entity implementation
*/

TellerEntity::TellerEntity(sim_event &ev)
{
  sim_trace(LEVEL1, "entering teller entity
constructor\n");
  TellerInit init_info;
  SIM_GET(TellerInit, init_info, ev);
  ID = init_info.ID;
  customer_service_generator =
init_info.customer_service_generator;
  vip_service_generator =
init_info.vip_service_generator;

# ifdef INSTRUMENTED
  char title[60];
  sprintf(title, "Teller %d Customers",ID);
  stats.customer_service_time = my_tally( title);
  sprintf(title, "Teller %d VIPs",ID);
  stats.VIP_service_time = my_tally( title);
# endif
}

void TellerEntity::body()
{
  sim_event ev;
  CustID newCustomer;
  VipID VIP;

  GetNextEvent( ev);
  while (ev.type() != REPORT_EV) {
    switch(ev.type()) {
      case CUSTOMER_EV:
    SIM_GET(CustID, newCustomer, ev);
        ServiceCustomer(newCustomer);
    break;
      case VIP_EV:
    SIM_GET(VipID, VIP, ev);
    ServiceVip(VIP);
    break;
      default:
    sim_error("unexpected event type %d from: %s\n",
ev.type(),
          ev.scheduled_by().name());
      }
    GetNextEvent( ev);
  }
# ifdef INSTRUMENTED
  WriteReport();
# endif
  sim_trace(LEVEL1, "Teller Terminated\n");
}

void TellerEntity::GetNextEvent(sim_event &ev)
{
    sim_schedule(::line_id[ID], 0.0, FREE_EV); // tell
line we are ready

    //wait for something
    sim_select(SIM_ANY, ev);
    if (ev == SIM_NO_EVENT)
    {
    sim_trace(LEVEL1, "Waiting for next customer or
VIP\n");
    sim_wait(ev);
    }
}

void TellerEntity::ServiceCustomer(CustID customer)
{
    sim_time cust_service_time,
service_time_remaining;
    sim_type_p VIP_p(VIP_EV);
    sim_event ev;

  sim_trace(LEVEL1, "handing customer service
event\n");

  sim_schedule(::line_id[ID], 0.0, BUSY_EV); //prevent
line from sending more

# ifdef INSTRUMENTED
  customer.Service_Started();
# endif

  cust_service_time = customer_service_generator-
>sample();
  service_time_remaining = cust_service_time;

  while ( service_time_remaining > 0.0)
  {
      service_time_remaining =
      sim_hold_for(service_time_remaining, VIP_p,
ev);
      if (service_time_remaining > 0.0) // interrupted
by VIP
      {
#     ifdef INSTRUMENTED
      stats.cust_interrupts++;
#       endif
      VipID VIP;
      SIM_GET(VipID, VIP, ev);
      ServiceVip( VIP);
      }
  }

# ifdef INSTRUMENTED
  stats.customer_service_time.update((sim_clock() -
customer.service_start));
# endif
}

void TellerEntity::ServiceVip(VipID )
{
  sim_time vip_service_time, remaining_service_time;
  sim_type_p newVIP_p(VIP_EV);
  VipID new_VIP;
  sim_event ev;

  sim_trace(LEVEL1, "handling vip customer service
event\n");

  sim_schedule(::line_id[ID], 0.0, BUSY_EV);
     //prevent line from sending new cust

  vip_service_time = vip_service_generator->sample();

  remaining_service_time = vip_service_time;
  while ( remaining_service_time > 0.0)
  {
      remaining_service_time =
      sim_hold_for( remaining_service_time, newVIP_p,
ev);
      if ( remaining_service_time > 0.0)   //
interrupted by another VIP
      {
      SIM_GET(VipID, new_VIP, ev);
      TurnAwayVip( new_VIP);
      }
  }

# ifdef INSTRUMENTED
  stats.VIP_service_time.update( vip_service_time);
# endif
}
```

```
void TellerEntity::TurnAwayVip(VipID)
{
#   ifdef INSTRUMENTED
    stats.vip_turnaway++;
#   endif
}

void TellerEntity::WriteReport()
{
  sim_trace(LEVEL1, "Writing Teller Report: Teller %d
\n", ID);

  sim_schedule(summary_id, 0.0,
REPORT_REPLY,SIM_PUT(TellerStats,stats));
}
```

APPENDIX D

SMALLTALK-80 CODE

```
SimulationObject subclass: #Test
    instanceVariableNames: ''
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Simulations'!

!Test methodsFor: 'list of tests'!
newtest4: num

|asim dummy block|
block _ [dummy < num ifTrue:
          [ | myToken |
          dummy _ dummy + 1.
          asim schedule: block at: 1.
          myToken _ self acquire: 1 ofResource:
'token'.
          self holdFor: 1.0; release: myToken]].
^Time millisecondsToRun:
[dummy _ 0 .
asim _ Simulation new.
asim produce: 1 of: 'token'.
asim schedule: block at: 1 .
asim startUp.
[asim proceed = nil] whileFalse
]! !

!Test methodsFor: 'list of tests'!
test1: num

|asim dummy|
^Time millisecondsToRun:
[dummy _ 0 .
asim _ Simulation new .
num timesRepeat:
 [asim schedule: [dummy _ dummy + 1]
        at: (Uniform from: 1 to: 1000)  next] .
asim startUp.
[asim proceed = nil] whileFalse .]! !

!Test methodsFor: 'list of tests'!
test2: num

|asim dummy block|
block _ [dummy < num ifTrue:
          [dummy _ dummy + 1.
          asim schedule: block at: 1]].
^Time millisecondsToRun:
[dummy _ 0 .
asim _ Simulation new.
asim schedule: block at: 1 .
asim startUp.
[asim proceed = nil] whileFalse
]! !

!Test methodsFor: 'list of tests'!
test3WithDelay: num

|asim|
^Time millisecondsToRun:
[asim _ Simulation new.
asim schedule: [Counter new countWithDelay: num] at: 1
.
asim startUp.
[asim proceed = nil] whileFalse
]! !

!Test methodsFor: 'list of tests'!
test3WithOneDelay: num

|asim|
^Time millisecondsToRun:
[asim _ Simulation new.
asim schedule: [Counter new countWithOneDelay: num]
at: 1 .
asim startUp.
[asim proceed = nil] whileFalse
]! !

!Test methodsFor: 'list of tests'!
test3WithOutDelay: num

|asim|
^Time millisecondsToRun:
[asim _ Simulation new.
asim schedule: [Counter new countWithOutDelay: num]
at: 1 .
asim startUp.
[asim proceed = nil] whileFalse
]! !

!Test methodsFor: 'list of tests'!
test4: num

|asim|
^Time millisecondsToRun:
[asim _ Simulation new .
 asim produce: 1 of: ' token' .
num timesRepeat:
 [asim schedule: [|token|
        token _ self acquire: 1 ofResource: ' token'
.
        self release: token ]
        at: 1] .
asim startUp.
[asim proceed = nil] whileFalse .]! !

!Test methodsFor: 'list of tests'!
test5: num

|asim inthand|
^Time millisecondsToRun:
[asim _ Simulation new .
inthand _ InterruptHandler new.
asim schedule: [ num timesRepeat: [inthand handleWith:
[].
    self holdFor: 100
    withHandler: inthand.]]
    at: 1 .
asim schedule: [ num timesRepeat: [self holdFor: 1 .
    inthand interrupt]]
    at: 1 .
asim startUp.
[asim proceed = nil] whileFalse .]! !



SimulationObject subclass: #CustomerObject
    instanceVariableNames: 'myGender bailOut '
    classVariableNames: ''
    poolDictionaries: ''
    category: 'SimBenchmark'!

!CustomerObject methodsFor: 'accessing'!
getGender
    ""

    ^myGender! !

!CustomerObject methodsFor: 'simulation control'!
getBestLine
    "return first available line, or otherwise line
with shortest queue"
```

```
    | best |
    best _ 1 .
    1 to: NUMLINES do: [:index |
      (self inquireFor: 1 ofResource: (index
printString))
        ifTrue: [^index].
      ((self numWaiting: index) < (self numWaiting:
best))
        ifTrue: [best _ index]
    ].
    ^best! !

!CustomerObject methodsFor: 'simulation control'!
goToThePotty
    "simulate going to the potty"

    | tolerance resname token wait |
    tolerance _ (Gamma mean: 2.0 var: 0.5)
improvedNext.
    resname _ 'pot' , (myGender printString).
    ((((Simulation active) provideResourceFor:
resname) returnPendingNum)
      > tolerance)
       ifTrue: [bailOut _ true]
       ifFalse:
         [token _ self acquire: 1 ofResource:
resname.
          (myGender = 0)
                ifTrue:  [wait _ (Gamma mean: 5.0
var: 1.0 ) next]
                ifFalse: [wait _ (Gamma mean: 3.0
var: 1.0 ) next].
          self holdFor: wait .
          self release: token]! !

!CustomerObject methodsFor: 'simulation control'!
numWaiting: index
    "return number waiting on resource index"

    | resource |
    resource _ (Simulation active)
provideResourceFor: (index printString).
    ^resource returnPendingNum! !

!CustomerObject methodsFor: 'simulation control'!
tasks
    "Customer subobject tasks"

    | banksim myline mystring myvipstring mytoken
servicetime starttime
      partialtime pottyinthand vipinthand |
    banksim _ Simulation active.
    pottyinthand _ InterruptHandler new
        handleWith: [self goToThePotty].
    mytoken _ nil.
    [mytoken = nil] whileTrue:
        [bailOut ifTrue: [^nil].
        banksim schedule: [pottyinthand interrupt]
after: ((Gamma mean: 3.0 var: 1.0) improvedNext ).
        myline _ self getBestLine.
        mystring _ myline printString.
        myvipstring _ 'vip' , mystring.
        mytoken _ self acquire: 1 ofResource:
mystring withHandler: pottyinthand].
    vipinthand _ (banksim vipHandler: myline)
        handleWith:
    [partialtime _ (banksim time) - starttime.
    servicetime _ servicetime - partialtime.
    self release: (self acquire: 1 ofResource:
myvipstring)].
    servicetime _ (Exponential mean: 10.0) next .
    starttime _ banksim time.
    (servicetime > 0.0) ifTrue: [self holdFor:
servicetime withHandler: vipinthand].
    self release: mytoken.! !

!CustomerObject methodsFor: 'initialization'!
initialize
    "initialize instance variables"

    myGender _ ((RAND next) + 0.5) truncated.
    bailOut _ false! !

CustomerObject subclass: #VIPObject
    instanceVariableNames: ''
    classVariableNames: ''
    poolDictionaries: ''
    category: 'SimBenchmark'!

!VIPObject methodsFor: 'simulation control'!
tasks
    "vip object tasks"

    |banksim myline mystring myvipstring mytoken
myviptoken|
    mytoken _ nil.
    banksim _ Simulation active.
    myline _ (RAND next * NUMLINES) truncated + 1 .
    mystring _ myline printString.
    myvipstring _ 'vip' , mystring.
    (self inquireFor: 1 ofResource: myvipstring)
        ifTrue:  [myviptoken _ self acquire: 1
ofResource: myvipstring]
        ifFalse: [^nil].
    (self inquireFor: 1 ofResource: mystring)
        ifTrue:  [mytoken _ self acquire: 1
ofResource: mystring]
        ifFalse: [(banksim vipHandler: myline)
interrupt].
    self holdFor: (Exponential mean: 7.0) next .
    self release: myviptoken.
    (mytoken = nil) ifFalse: [self release: mytoken]!
!

Simulation subclass: #BankSimulation
    instanceVariableNames: 'vipIntHandlers '
    classVariableNames: ''
    poolDictionaries: ''
    category: 'SimBenchmark'!

!BankSimulation methodsFor: 'initialization'!
defineArrivalSchedule
    "Bank simulation subclass provides the simulation
objects"

    self scheduleArrivalOf: CustomerObject
                accordingTo:  (Exponential mean:
2.0);
        scheduleArrivalOf: VIPObject
                accordingTo:  (Exponential mean:
2.0)! !

!BankSimulation methodsFor: 'initialization'!
defineResources
    "Bank simulation resource initialization"

    |indexstring|
    1 to: NUMLINES do:
      [:index |
        indexstring _ index printString.
        self produce: 1 of: indexstring.
        self produce: 1 of: ('vip' , indexstring)].
    self produce: 4 of: 'pot' , 0 printString;
        produce: 4 of: 'pot' , 1 printString.! !

!BankSimulation methodsFor: 'initialization'!
initialize
    "additional initializations for Bank Simulation"

    super initialize.
    Transcript show: '      Bank Sim Running
'.
     Smalltalk at: #RAND put: (Random new).
    Smalltalk at: #NUMLINES put: 3 .
    vipIntHandlers _ Array new: NUMLINES.
    1 to: NUMLINES do:
        [:index | vipIntHandlers at: index put:
(InterruptHandler new)].! !

!BankSimulation methodsFor: 'accessing'!
vipHandler: num
    "return vip int handler for line num"
```

```
    ^vipIntHandlers at: num! !

Object subclass: #RunBankSim
    instanceVariableNames: ''
    classVariableNames: ''
    poolDictionaries: ''
    category: 'SimBenchmark'!

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
"!

RunBankSim class
    instanceVariableNames: ''!

!RunBankSim class methodsFor: 'all'!
time: value
    "comment stating purpose of message"

    | banksim |
    banksim _ BankSimulation new startUp.
    [banksim time < value] whileTrue: [banksim
proceed]! !
```

APPENDIX E

MOOSE CODE

```
////////// Test 1:

#include "/va/jil/event/event.h"
#include <stdlib.h>
#include <ACG.h>
#include <Uniform.h>
#include <stream.h>

ACG gen;

Uniform r(0.0, 1000.0, &gen);

class foo {
   public:
     void bar();
};

foo *f;

class foobarEvent : public Event {
   public:
     foobarEvent(foo *f) : F(f) {}
   private:
     foo *F;
     virtual void start() { F->bar(); }
};

void foo::bar() {
}

main(int argc, char *argv[]) {
     int n = atoi(argv[1]);
     f = new foo;
     init_simulation(n);
     while (n--)
       EVENT(foobarEvent, (f), r());
     run_simulation();
     print_event_stats();
     HALT();
}


////////// Test 2:

#include "/va/jil/event/event.h"
#include <stdlib.h>

class foo {
   public:
     void bar(int);
};

foo *f;

class foobarEvent : public Event {
   public:
     foobarEvent(int N) : n(N) {}
   private:
     int n;
     void start() { f->bar(n); }
};

void foo::bar(int n) {
     if (--n > 0)
       EVENT(foobarEvent, (n), 0.0);
}

main(int argc, char *argv[]) {
     int n = atoi(argv[1]);
     f = new foo;
     init_simulation(n);
     EVENT(foobarEvent, (n), 0.0);
     run_simulation();
     print_event_stats();
}

////////// Test 3a:

#include "/va/jil/event/event.h"
#include <stdlib.h>

class foo {
   public:
     void bar(int);
};

foo *f;

class foobarEvent : public Event {
   public:
     foobarEvent(int N) : n(N) {}
   private:
     int n;
     void start() { f->bar(n); }
};

void foo::bar(int n) {
     if (--n > 0)
       f->bar(n);
}

main(int argc, char *argv[]) {
     int n = atoi(argv[1]);
     f = new foo;
     init_simulation(n);
     EVENT(foobarEvent, (n), 1.0);
     run_simulation();
     print_event_stats();
}


////////// Test 3b:

#include "/va/jil/event/event.h"
#include <stdlib.h>

class foo {
   public:
     void bar(int);
};

foo *f;

class foobarEvent : public Event {
   public:
     foobarEvent(int N) : n(N) {}
   private:
     int n;
     void start() { f->bar(n); }
};

void foo::bar(int n) {
     event_delay(1.0);
     if (--n > 0)
       f->bar(n);
}

main(int argc, char *argv[]) {
     int n = atoi(argv[1]);
     f = new foo;
     init_simulation(n);
     EVENT(foobarEvent, (n), 1.0);
     run_simulation();
     print_event_stats();
////////// Test 4:
```

```
#include "/va/jil/event/event.h"
#include "/va/jil/event/resource.h"
#include <stdlib.h>

class CustomerObj {
   public:
     void Run(int);
};

CustomerObj *Cust;
Resource *Res;

class RunEvent : public Event {
   public:
     RunEvent(int n) : N(n) {}
     void start() { Cust->Run(N); }
     int N;
};

void CustomerObj::Run(int n) {
     if (n > 1)
       EVENT(RunEvent, (n - 1), 0.0);
     Res->give(1);
     event_delay(1.0);
     Res->take_back(1);
}

main(int argc, char *argv[]) {
     int i = atoi(argv[1]);
     init_simulation();
     Cust = new CustomerObj;
     Res = new Resource;
     Res->create(1);
     EVENT(RunEvent, (i), 0.0);
     run_simulation();
     print_event_stats();
}


////////// Test 5:

#include "/va/jil/event/event.h"
#include "/va/jil/event/interrupt.h"
#include <stdlib.h>

class foo {
   public:
     void Tripper(int);
     void Trippee(int);
};

foo *f;

EID Trippee_eid;

DECLARE_INTERRUPT(Trip_Interrupt);

void foo::Trippee(int n) {
     for (int i = 0; i < n; i++) {
       TRAP event_delay(100.0);
       HANDLE(Trip_Interrupt);
       END_TRAP;
     }
}

void foo::Tripper(int n) {
     for (int i = 0; i < n; i++) {
       event_delay(1.0);
       event_interrupt(Trippee_eid, Trip_Interrupt);
     }
}

class fooTrippeeEvent : public Event {
   public:
     fooTrippeeEvent(foo *f, int n) : F(f), N(n) {}
   private:
     foo *F;
     int N;
     void start() { F->Trippee(N); }
};

class fooTripperEvent : public Event {
   public:
     fooTripperEvent(foo *f, int n) : F(f), N(n) {}
   private:
     foo *F;
     int N;
     void start() { F->Tripper(N); }
};

main(int argc, char *argv[]) {
     int n = atoi(argv[1]);
     f = new foo;
     init_simulation();
     EVENT(fooTripperEvent, (f,n), 0.0);
     EVENT(fooTrippeeEvent, (f,n), 0.0);
     run_simulation();
     print_event_stats();
}


////////// Sim (the bank simulation):

#include "/va/jil/event/event.h"
#include "/va/jil/event/interrupt.h"
#include "/va/jil/event/resource.h"
#include <MLCG.h>
#include <Erlang.h>
#include <NegativeExpntl.h>
#include <DiscreteUniform.h>
#include <limits.h>
#include <stream.h>

MLCG randomgen;

DiscreteUniform rand_gender(0, 1, &randomgen);

Erlang rand_nature(1.0, 1.0, &randomgen);
// change params later to meanNatureCallsTime and
varianceNatureCallsTime

Erlang rand_line(1.0, 1.0, &randomgen);
// change params later to meanLineTolerance and
varianceLineTolerance

Erlang rand_frest(5.0, 8.0, &randomgen);

Erlang rand_mrest(3.0, 6.0, &randomgen);

DiscreteUniform rand_numlines(0, 1, &randomgen);
// change max to numLines - 1

Erlang rand_serve(1.0, 1.0, &randomgen);
// change mean later to meanServiceTime,
varianceServiceTime;

NegativeExpntl rand_custarrive(1.0, &randomgen);
// change mean later to meanInterArriveTime

NegativeExpntl rand_viparrive(1.0, &randomgen);
// change mean later to meanVIPInterArriveTime


enum Gender {Female, Male};

class LineObj : public Resource {
   public:
     class CustomerObj *serving;
     EID service_eid;
     void ServeCust(class CustomerObj *cust);
};

typedef LineObj *LineObjP;

class ServeCustEvent : public Event {
   public:
     ServeCustEvent(LineObj *l, CustomerObj *c) :
L(l), C(c) {}
     void start() { L->ServeCust(C); }
   private:
     LineObj *L;
     CustomerObj *C;
};
```

```
typedef Resource RestRoom;

class CustomerObj {
   public:
     Gender myGender;
     CustomerObj();
     virtual void GetOnLine();
     void VisitFacilities();
     virtual int isVIP() { return 0; }
   private:
     LineObj *FindBestLine();
};

class GetOnLineEvent : public Event {
   public:
     GetOnLineEvent(CustomerObj *c) : C(c) {}
     void start() { C->GetOnLine(); }
   private:
     CustomerObj *C;
};

class VIPObj : public CustomerObj {
   public:
     void GetOnLine();
     int isVIP() { return 1; }
};

class CustGenerator : public Event {
   protected:
     NegativeExpntl *interArrive;
     virtual void newCust() { CustomerObj *co = new
CustomerObj; }
   public:
     CustGenerator() : interArrive(&rand_custarrive)
{}
     void start();
};

class VIPGenerator : public CustGenerator {
   protected:
     virtual void newCust() { VIPObj *vo = new VIPObj;
}
   public:
     VIPGenerator() { interArrive = &rand_viparrive; }
};

CustomerObj::CustomerObj() {
     myGender = rand_gender();
     EVENT(GetOnLineEvent, (this), 0.0);
}

DECLARE_INTERRUPT(Nature_Calls_Interrupt);

class NatureCallsEvent : public Event {
   public:
     NatureCallsEvent() { eid = Current_EID(); }
     void start() { event_interrupt(eid,
Nature_Calls_Interrupt); }
   private:
     EID eid;
};

double hoursToRun;
double meanInterArriveTime;
double meanVIPInterArriveTime;
double meanNatureCallsTime;
double varianceNatureCallsTime;
double meanLineTolerance;
double varianceLineTolerance;
double meanServiceTime;
double varianceServiceTime;
int numLines;
LineObj **allLines;
RestRoom *restRooms[2];
long seed;

int numCusts = 0;
int numVips = 0;
int numCustLeaving = 0;
int numMadVips = 0;
int numNatureCalls = 0;
int numVipInterrupts = 0;

Sim_Time totalTimeUntilServed = 0.0;
Sim_Time totalCustServiceTime = 0.0;

void CustomerObj::GetOnLine() {
     LineObj *myLine;
     Sim_Time timeTillNatureCalls;
     // numCusts++;
     EID timeout;
     Sim_Time time = simulation_time();

     for(;;) {
       for (;;) {
            timeTillNatureCalls = rand_nature();
            myLine = FindBestLine();
            timeout = EVENT(NatureCallsEvent, (),
timeTillNatureCalls);
            TRAP {
              myLine->give(1);
              break;
            } HANDLE(Nature_Calls_Interrupt)
              VisitFacilities();
            END_TRAP;
       }
       totalTimeUntilServed += simulation_time() -
time;
       event_terminate(timeout);
       // time = simulation_time();
       myLine->ServeCust(this);
       // totalCustServiceTime += simulation_time() -
time;
       myLine->take_back(1);
       break;
     }
     delete this;
}

void CustomerObj::VisitFacilities() {
     RestRoom *restRoom;
     int restRoomLineTolerance;
     // numNatureCalls++;
     restRoom = restRooms[myGender];
     restRoomLineTolerance = int(rand_line() + 0.5);
     if (restRoom->num_pending() >
restRoomLineTolerance) {
       delete this;
       // numCustLeaving++;
       event_terminate();
     } else {
       restRoom->give(1);
       event_delay((myGender == Female)? rand_frest():
rand_mrest());
       restRoom->take_back(1);
     }
}

LineObj *CustomerObj::FindBestLine() {
     LineObj *line, *bestLine;
     int length, bestLength;

     bestLength = INT_MAX;
     for (int i = 0; i < numLines; i++) {
       line = allLines[i];
       length = line->num_pending();
       if (line->available() == 0) length++;
       if (length < bestLength) {
            bestLength = length;
            bestLine = line;
       }
     }
     return bestLine;
}

DECLARE_INTERRUPT(VIP_Arrives_Interrupt);
DECLARE_INTERRUPT(VIP_Leaves_Interrupt);

void VIPObj::GetOnLine() {
     LineObj *line;
     CustomerObj *oldCust;
     // numVips++;
     int nline = int(rand_numlines());
     line = allLines[nline];
     oldCust = line->serving;
```

```
      EID oldservice = line->service_eid;
      if (oldCust) {
        if (oldCust->isVIP()) {
            // numMadVips++;
            return;
        }
        // numVipInterrupts++;
        event_interrupt(oldservice,
VIP_Arrives_Interrupt);
      } else
        line->give(1);
      line->ServeCust(this);
      if (oldCust)
        event_interrupt(oldservice,
VIP_Leaves_Interrupt);
      else
        line->take_back(1);
      delete this;
}

void LineObj::ServeCust(CustomerObj *cust) {
      Sim_Time svcTime, startTime;

      svcTime = rand_serve();
      for(;;) {
        service_eid = Current_EID();
        startTime = simulation_time();
        serving = cust;
        if (svcTime <= 0.0)
            break;
        TRAP {
            event_delay(svcTime);
            serving = 0;
            break;
        } HANDLE(VIP_Arrives_Interrupt) {
            svcTime -= simulation_time() - startTime;
            TRAP event_suspend();
            HANDLE(VIP_Leaves_Interrupt);
            END_TRAP;
        } END_TRAP;
      }
}

void CustGenerator::start() {
      CustomerObj *customer;
      Sim_Time waitTime;

      for (;;) {
        waitTime = (*interArrive)();
        event_delay(waitTime);
        newCust();
      }
}

class EndSim : public Event {
   public:
      void start() {
        print_event_stats();
        HALT();
      }
};

main() {
      restRooms[Female] = new RestRoom;
      restRooms[Male] = new RestRoom;
      restRooms[Female]->create(4);
      restRooms[Male]->create(4);
      cout << "C++ Simulation 'Lines and Rest Rooms'
starting--\n";
      cout << "What is the mean customer interarrival
time in minutes? ";
      cin >> meanInterArriveTime;
      rand_custarrive.mean(meanInterArriveTime);
      cout << "What is the mean VIP interarrival time
in minutes? ";
      cin >> meanVIPInterArriveTime;
      rand_viparrive.mean(meanVIPInterArriveTime);
      cout << "What is the mean service time in
minutes? ";
      cin >> meanServiceTime;
      rand_serve.mean(meanServiceTime);
      cout << "What is the variance of the service
time? ";
      cin >> varianceServiceTime;
      rand_serve.variance(varianceServiceTime);
      cout << "What is the mean time in minutes till
'Nature Calls'? ";
      cin >> meanNatureCallsTime;
      rand_nature.mean(meanNatureCallsTime);
      cout << "What is the variance? ";
      cin >> varianceNatureCallsTime;
      rand_nature.variance(varianceNatureCallsTime);
      cout << "What is the mean restroom line length
tolerance? ";
      cin >> meanLineTolerance;
      rand_line.mean(meanLineTolerance);
      cout << "What is the variance? ";
      cin >> varianceLineTolerance;
      rand_line.variance(varianceLineTolerance);
      cout << "How many lines are there? ";
      cin >> numLines;
      rand_numlines.high(numLines-1);
      cout << "How many hours should the simulation
run? ";
      cin >> hoursToRun;
      cout << "Random Seed? ";
      cin >> seed;

      cout << "\nMean Interarrive Time: " <<
meanInterArriveTime << "\n";
      cout << "Mean VIP Interarrive Time: " <<
meanVIPInterArriveTime
<< "\n";
      cout << "Mean Service Time: " <<  meanServiceTime
<< "\n";
      cout << "Variance Service Time: " <<
varianceServiceTime << "\n";
      cout << "Mean 'Nature Calls' Time: " <<
meanNatureCallsTime << "\n";
      cout << "Variance 'Nature Calls' Time: " <<
varianceNatureCallsTime << "\n";
      cout << "Mean line length tolerance: " <<
meanLineTolerance << "\n";
      cout << "Variance line length tolerance: " <<
varianceLineTolerance << "\n";
      cout << "Number of lines: " <<  numLines << "\n";
      cout << "Hours to run: " <<  hoursToRun << "\n";

      allLines = new LineObjP[numLines];
      for (int k = 0; k < numLines; k++) {
        LineObj *line = new LineObj;
        line->create(1);
        allLines[k] = line;
      }

      randomgen.reseed(seed, seed);

      init_simulation(4000, 2000);

      EVENT(EndSim,, hoursToRun * 60.0);
      EVENT(CustGenerator, (), 0.0);
      EVENT(VIPGenerator, (), 0.0);

      run_simulation();
      /*
      cout << numCusts << " customers arrived.\n";
      cout << numVips << " VIPs arrived.\n";
      cout << numMadVips << " VIPs left angrily.\n";
      cout << "There were " << numNatureCalls << "
visits to the
restrooms.\n";
      cout << numVipInterrupts << " customers were
interrupted by VIPs.\n";
      cout << numCustLeaving << " customers left
without being served.\n";
      cout << "The average customer service time was "
<<
        totalCustServiceTime / (numCusts -
numCustLeaving) << "\n";
      cout << "The average customer wait time was " <<
        totalTimeUntilServed / (numCusts -
numCustLeaving) << "\n";
```

```
     for (k = 0; k < numLines; k++) {
       cout << "\nThe statistics for the length of
line #" << k+1 <<
" are:\n";
       allLines[k]->report_stats();
     }

     cout << "\nThe statistics for the length of the
mens room line
are:\n";
     restRooms[Male]->report_stats();

     cout << "\nThe statistics for the length of the
ladies room line
are:\n";
     restRooms[Female]->report_stats();
     */
     print_event_stats();
}
```

APPENDIX F

ERIC CODE

```
;;; test1

(define-class Nothing (:parents Something))
(ask Nothing when receiving (bar) nil)

 (defun test1 (n)
        (declare (type fixnum n))
        (ask Nothing make instance foo)
        (dotimes (i n)
                (declare (type fixnum i))
                (ask clock to schedule !foo to (bar)
at !(random 1000)))
        (ask clock run to completion))

(defvar *Nothings*)

(defun test1a-internal (n)
  (declare (type fixnum n))
  (dotimes (i n)
           (declare (type fixnum i))
           (let ((x (svref *nothings* i)))
             (ask clock to schedule !x to (bar) at
!(random 1000))))
  (ask clock run to completion))

(defun test1a (n)
  (let ((*nothings* (make-array n)))
    (dotimes (i n)
             (setf (aref *nothings* i) (ask Nothing
make instance foo)))
    (time (test1a-internal n))))

;;;test2

 (define-class Nothing (:parents Something))
 (ask Nothing when receiving (bar >m)
     (if (> m 1) (ask clock to schedule !self to (bar
!(decf m)) at 0)))

(defun nextbaz (n)
  (declare (type fixnum n))
  (if (> n 1)
      (let ((n-1 (1- n)))
    (declare (type fixnum n-1))
    (ask clock to schedule !foo to (baz !n-1) at
0))))

(ask Nothing when receiving (baz >n)
     (nextbaz n))

 (defun test2 (n)
    (declare (type fixnum n))
    (ask clock set your simtime to 0)
        (ask Nothing make instance foo)
        (ask clock to schedule !foo to (bar !n) at 0)
    (ask clock run to completion))

;;;test3a

(define-class Nothing (:parents Something))

(pcl:defmethod bar1 ((self Nothing) n)
  (when (plusp (decf n))
    (bar1 self n)))

(ask Nothing when receiving (bar >n)
     (decf n)
     (bar1 self n))

(defun test3a (n)
  (declare (type fixnum n))
  (ask clock set your simtime to 0)
  (ask Nothing make instance foo)
  (ask clock to schedule !foo to (bar !n) at 0)
  (ask clock run to completion))

;;;test3b

(define-class Nothing (:parents Something))

(pcl:defmethod bar1 ((self Nothing) n)
           (declare (type fixnum n))
           (when (plusp (decf n))
             (ask clock to schedule !self to
(complete-bar !n) in 1 second)))

(ask Nothing when receiving (bar >n)
     (bar1 self n))

(ask Nothing when receiving (complete-bar >n)
     (bar1 self n))

(defun test3b (n)
  (declare (type fixnum n))
  (ask clock set your simtime to 0)
  (ask Nothing make instance foo)
  (ask clock to schedule !foo to (bar !n) at 0)
  (ask clock run to completion))

;;;test4

(define-class Nothing (:parents Something))

(ask Nothing when receiving (run >n)
     (when (> n 1)
       (ask clock to schedule !self to (run !(1- n))
in 0 seconds))
     (ask res give 1 and ask !self run2 !n))

(ask Nothing when receiving (run2 >n)
     (ask clock to schedule !self to (run3 !n) in 10
seconds))

(ask Nothing when receiving (run3 >n)
     (ask res take-back 1))

(defun test4 (n)
  (declare (type fixnum n))
  (ask resource make instance res)
  (ask res create 1)
  (ask Nothing make instance cust)
  (ask clock to schedule !cust to (run !n) in 0
seconds)
  (ask clock run to completion))
```

DISTRIBUTION LIST

INTERNAL

A010

R. D. Haggarty
B.M. Horowitz

A030

R. W. Jacobus
H. W. Sorenson
L. M. Thomas

D010

E. J. Ferrari
D. D. Neuman

D040

D. I. Buckley
G. J. Koehr
J. C. Naylor, Jr.

D050

R. A. McCown
E. A. Palo
E. N. Skoog

D060

J. K. DeRosa
C. H. Nordstrom, Jr.

D070

E. H. Bensley (10)
A. L. Buchanan
J. A. Clapp
M. P. Galin
J. H. James
C. E. Kalish
D. A. MacQueen, Jr.
M. A. Makhlouf
T. F. Saunders

D071

J. M. Apicco
D. E. Currie
R. G. Howe
K. L. Lennon
J. A. Maurer
M. L. Morgillo
R. Platcow
R. B. Quanrud
J. S. Whalley
E. C. Wigfield
C. Y. Young

D072

D. Amano
R. A. Birtwell
J. C. Broderick
M. J. Brooks
J. R. Cherniack
J. R. Cottrell, Jr.
P. A. Dallas
D. A. Drake
R. I. Eachus
S. C. Ernst
D. A. Franciskovich
J. A. Francoeur
R. F. Furey-Deffely
V. T. Giddings (10)
W. M. Hornish
J. A. Houchens
P. C. Krupp
J. S. Lambe
R. J. Lesch
M. R. Lord
D. S. Lyons
R. W. Noel
W. F. Paton
P. J. Pelsinski
B. C. Robinson
M.E. Rothberg
A. Sateriale

P. D. Smith
P. R. Smith
M. F. Spears
S. J. Stella
K. E. Styles
L. A. Sun
S. W. Tavan
T. M. Wheeler
A. M. Willhite

D073

C. E. Baker
B. J. Bakis
B. Cui
M. T. Drozd
J. Finkel
S. R. Friedman
J. R. Knobel
M. T. Owens
T. J. Reale, Jr.
T. B. Rice
T. C. Royer

D074

T. K. Backman
P. A. Brown
W. C. Carter
L. P. Costa
S. I. Frank
G. M. Friedman
J. Gates
J. A. Gunter
M. Hazle
L. J. Holtzblatt
A. L. Kosmala
C. Loizides
R. A. Martin
R. J. McGue
F. R. Murphy
R. S. Popp
C. K. Reid
R. D. Rhode

D080

R. W. Bush
C. H. Gager
S. M. Newman

D090

L. S. Metzger
S. J. Pomponi

F044

M. A. Fabrizi
I. Frolow

F084

P. T. R. Wang

G010

V. A. DeMarines

G030

N. E. Bolen
R. F. Nesbit

G110

H. A. Bayard
R. C. Labonte'
E. L. Lafferty
S. D. Litvintchouk
M. E. Nadel
M. J. Prelle
P. S. Tasker

G111

J. D. Jacobs
J. I. Leivent
M. T. Maybury
A. M. Wollrath

G115

B. M. Thuraisingham

G117

T. J. Brando
H. G. Goldman
P. J. Guay
D. M. Johnson
L. G. Monk
J. T. Trostle
R. J. Watro (20)
A. M. Wollrath

J070

J. G. Sprung

J080

H. Carpenter
D. H. Gill
R. P. Granato
F. X. Maginnis
A. Sears
J. K. Summers

J082

H. Cohen

W153

M. L. Kahn

W156

W. P. Niedringhaus